Contents

Overview

Commands

Hints

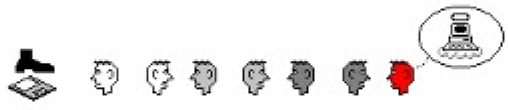
Future developments

Use the scroll bar to see entries not currently visible in the Help window.

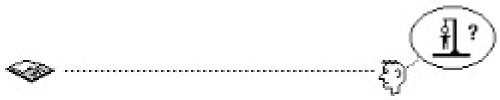
To learn how to use Help, press F1 or choose Using Help from the Help menu

Overview: Purpose

Most programs do not show the contents of a file - they interpret it.



MicroFile displays the contents in their **actual** state.



For example, is your sensitive data encrypted or only the program that requires a password ?

Why will word processors not read each others files ?

Can you identify redundant files ?

This program will not answer these - but it will help **you** to find out.

With MicroFile you can **see** if your data is encrypted.

Word processors only display the text from their data files.

They never show the codes used for font name, size etc. which are also stored there.

As these codes **differ** between word processors, the files are never **directly** transferrable.

Unless your word processor can **translate** that format, the document is inaccessible.

MicroFile can strip text out of **any** file.

It can avoid the delay while someone sends you another copy in a format you both can use.

It does not matter what type of file it is - whether it is a document, data file or even a program - see **extracting text**.

In the past you may have tried a demonstration version of a program and later deleted it.

However, many programs install files into your Windows or DOS directories - without informing you !

Using MicroFile, you can examine the contents of a potentially redundant file and then make an informed decision on whether it can safely be deleted.

(Copy or rename it - never delete it until you are certain it is not needed - you might be wrong !)

Also, consider a file with the following two records

```
Fred  Bloggs      10 High Street      Newcastle upon Tyne
Freda Bloggs   10 High Street   Newcastle upon Tyne
```

The format appears identical, but one is 65 characters using spaces, the other 52 using **tab** positions.

A program would have problems with this file even though they pass a visual inspection.

The display is not restricted to a fixed number of characters, consider

fixed 16 char displays	variable width

`first 14 charsse`
`cond 14 numsthir`
`d 14 chars`
	`first 14 chars`
	`second 14 nums`
	`third 14 chars`

`jackishonanevila`
`nton`
	`jacki`
	`shona`
	`nevil`
	`anton`

For program developers, it can aid in analyzing and debugging problems, as well as assisting in analysis of undocumented file structures.

MicroFile can display files using three international formats: **ANSI**, **ASCII** and **EBCDIC**, as well as the **OEM** character set.

There is a choice of fonts and sizes - giving a variety of characters per line and lines per screen.

Characters per line variable from 1 to full screen width.

Sideways scrolling for text records.

Displays can be in hex and character, hex only, character only or text record/word wrap formats.

File position can be in hex or decimal, starting from 0 or 1 - with the option to display the address or not.

A search facility - forward/back, from start/end, hex or character (case sensitive if required), and also, not equal to.

Facilities for screen prints, copy to clipboard or to disk are all available.

Go to function - either absolute or relative (in hex or decimal).

Define your own character set - **ASCII+** - standard **ASCII** plus some **ANSI** characters e.g. £, ©

Vary background/foreground colours, choose your own **unprintable** characters, text delimiters.

Display repeating lines as "nn lines as above" instead of wading through page after page.

All settings are retained between sessions.

Calculator functions to convert integer and floating point numbers to decimal, date formats, file pointers **and** navigate the file using them.

It can access files up to 2.1 Gigabytes in size.

It can be used to help determine the file type - regardless of it's file extension.

For example an EXE file always starts with the characters MZ - see **file identifiers**.

Screen Displays

The default screen display is to show file position, contents in **hex** and **character** format :-

Address Hex Char

0000 0001 3031 3233 3435 3637 3839 4142 4344 +0123456789ABCD+
0000 000E 3031 3233 3435 3637 3839 4142 4344 +0123456789ABCD+

Displaying the address is optional, and can be in hex or decimal format, starting at 0 or 1.

For example, the same lines could be displayed :-

0 3031 3233 3435 3637 3839
4142 4344 +0123456789ABCD+
13 3031 3233 3435 3637 3839
4142 4344 +0123456789ABCD+

or with the characters over the hex values

0 0 1 2 3 4 5 6 7 8
9 A B C D
3031 3233 3435 3637 3839
4142 4344
13 0 1 2 3 4 5 6 7 8
9 A B C D
3031 3233 3435 3637 3839
4142 4344

By default, the characters are displayed with **+...+** as **delimiters** to indicate where the characters start and stop, but these can be changed to any value as required e.g. { } [ ] or even spaces.

The following screen displays are supported :-

Hex and Char the file contents are shown in **hex** and **character** formats alongside each other.

Char and Hex as Hex_and_Char but with the characters placed above the hex values.

Hex only **hex** format only

Char only **character** only

Text - Records **text** format, but displaying only strings of characters that match the character set in use, the **delimiters** and other criteria specified via the Options/Records format - see **extracting text**.

Unsupported characters are considered invalid.

Sideways scrolling is available to read records that are too large to fit on a line.

Text - Word wrap Text records can be word wrapped if required.

Base 64 decode If UU or MIME encoded sections are contained in the file then this switches decoding ON, OFF or automatic on text displays.

For full details see **Menu and Toolbar Index**

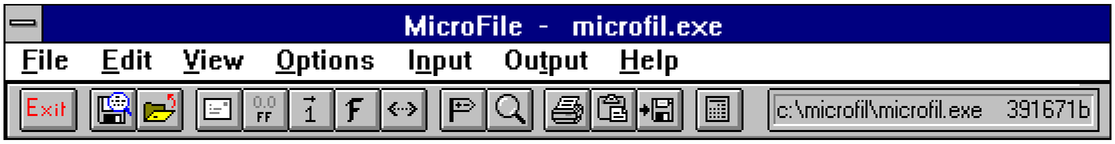
The display adjusts itself to the maximum characters per line and lines per screen that can be supported within the current screen definition.

With hex displays it will always pick the largest set of 8 characters that will fit - usually 16 or 24.

This can be varied from a single character to full screen width if this aids interpretation of the data.

Menu and Toolbar Index

Click the part that you want to know more about.



File

New Select a new file.

If the file is a compressed file (by Microsoft compress.exe) then the program will automatically ask if you wish to view this in it's actual state or in uncompressed form.

This is done in flight and without any intermediate files being used; **all** program functions (except editing) are fully supported in both modes.

These files are usually indicated by file extensions nn_ and are often found on Visual Basic application disks.

Please note - uncompressed, not expanded - there are many instances of "compressed" files which take up more space than the original !

This can happen if an image file which is already stored in a compressed form is "compressed" a second time.

There are options on the file dialogue screen to show hidden/system files or directories.

Note - the program cannot access hidden directories - it merely highlights

that one exists - you would have to change it's attributes to access it.

Open Re-open a file (defaults to NEW if no previous file selected).

Close Close an open file (useful if file-sharing).

Print Copy the screen to the default printer.

Full name A list of up to 8 previously viewed files is maintained - this option switches

the list between full hierarchic and file name only.

When displaying file names only, if two or more are identical then they will automatically be expanded to full hierarchic form.

Exit Exit the program.

Edit

Copy to Clipboard Copy the current screen to the clipboard.

In text displays, it copies the entire content of every record on screen to the clipboard, whether fully visible on screen or not.

Note - it does not empty the clipboard after use.

Large text records can tie up severe amounts of memory e.g.

17 lines x 1200 bytes requires a 20k buffer

Flush the buffer after use !

(after pasting, simply cut a couple of characters back into the clipboard)

Copy to Disk

Normal processes the entire file, re-directing the screen output to a disk file.

When used in extracting text out of word processing documents the new file should occupy less space than the original.

If you find the new file out-stripping the existing file position, then you may find you are not copying the format you expected !

Base 64 Decode searches for encoded section, decodes this and outputs to file

UU/MIME encode processes whole file, applying the required encoding and outputs to file

The progress of the copy is shown on screen and there is an option to abandon the copy if required.

The display shows the type of processing, current file size, original file position, the display being copied and the type of output file (PC, Mac, Unix etc.)

The output is written to a temporary workfile in the target directory called

MFhhmmss.tmp

where hhmmss is the time of day that the file was created

Only when the process completes successfully is the file renamed with the required title (after deleting any existing file of the same name).

If Cancel is selected at any time during the process, the program will stop copying to the workfile and prompt to see if it should be deleted.

Goto Go to a position within the file. This can be absolute i.e. byte 200 or relative +10, -20 etc. and can be specified in **hex** or decimal.

Find Search for a specified string :- forwards/backwards from start/end hex/character case sensitive not equal to

Searches can be repeated using **Find Next** or **Find Previous**.

Replace not implemented yet (May 1996) - see **future developments**

Edit not implemented yet (May 1996) - see **future developments**

View

Address Switches the display address on or off.

Decimal/Hex Display address in decimal or **hex**

Base 0/1 Set the address of the first byte in the file to 0 or 1.

Columns Used in normal mode to vary the number of characters displayed. In text mode it is used to move the window left and right over the records.

Font Varies the fixed pitch font - Fixedsys 9, Courier New 9, 8.25, 7.6 and 6.75.

The display is adjusted to provide maximum characters per line and lines per screen, according to the type of display (VGA, SVGA etc.)

If the display involves hex, then by default, the program will select the largest set of 8 bytes that can fit on the screen - this is usually 16 or 24.

The display can then be changed (using Columns) to vary the characters per line between 1 and the maximum the screen can contain.

Hex & Char Hexadecimal on left of screen; character display on right
Char & Hex Characters are displayed on the line above the corresponding hex value

Hex/Char only display either hex or character only

Text records see **extracting TEXT** for full details.

Text word wrap as Text records but wrapping each record to fit on the screen

Base 64 decode determines how any embedded UU or MIME encoded **sections** will be displayed.

Any screen line containing a UU or MIME section has the characters **u**, **U, m**, or **M** appended to it to indicate what type of encoding and whether it extends over part or all of the screen line

On when the file is opened it is searched to see if it contains any encoded sections.

as the encoding process results in 4 characters for every 3 in the original

file, so hex or character displays are padded out to reflect their actual position within the file i.e.

```
U O n  c ·  e   u ·  p o  n ·   a   ·  t i  m ·  e
  4F6E 63·· 6520 75··  706F 6E·· 2061 20··  7469 6D·· 65
```

text displays are shown without any padding.

Off no file searching or decoding is done - the file contents are displayed in their actual state.

```
T 2  5 j  Z S  B 1   c G  9 u  I G  E g   d G  l t  Z
5432 356A 5A53 4231  6347 3975 4947 4567  6447 6C74 5A
```

Auto on text no file searching is done; any string of characters which fail the **extracting TEXT** tests are tested to see if they may be encoded.

If they are, then any invalid characters in the decoded string are replaced

by spaces and the decoded string is re-processed to see if it contains text.

Whilst this is not particularly accurate it does identify that encoded strings

have been encountered and Base 64 Decode can be switched ON.

Options

Background
Foreground

The colour of the backround and foreground (text) can be changed as required (not both to the same colour).

As above

With this on, any lines that repeat will be displayed as "nn lines as above"

It is not uncommon to get 1000 chars repeated in uncompressed bitmaps.

Delimiters

By default, the text display uses **+**....**+** to indicate the start and end of characters, for example

```
+a normal line followed by a dummy+
+line with trailing spaces                +
```

This option allows any other keyboard character to be used.

Unprintable

Codes that have no direct visual equivalent (printer controls etc) are displayed as Ž (hex 9F, decimal 159).

This option allows other characters to be used instead, excluding number and alphabetic characters.

Calculator

hex translation - see **calculator** for full details.

signed/unsigned integers (byte, word, long and double)
currency
32/64/80 bit floating point
binary (bit display)
date and time
file pointers
Low/High or **High/Low** byte order

Base date

Select which base date should be used for when interpreting dates.

There are options to translate word or long word numbers as days or seconds since the base date.

DOS uses a long word **L/H** to record seconds since midnight 1Jan 1970;

MAC uses a long word **H/L** to record seconds since midnight 1 Jan 1904

The 64 bit floating point numbers are interpreted as

integer part	days since base date
fractional part	seconds since midnight

The more common base date conventions are shown, and there is an option to select any base in the range 0-9999 A.D.

Note - all date calculations are based on the Gregorian calendar.
(see **future developments**)

Input/Output

Character Set This defines which character set/translation table should be used when reading the file in and also when copying the file to disk.
For further details see **character set**.

Record formats For a string to qualify as meaningful text, there are several tests that have

to be passed - for full details see **extracting text**.

Help

Contents Invokes the program you are currently looking at.

Tooltips Sets the time delay for the tooltips facility.
(at present these cover toolbar items and base date functions)

Licence details Displays the current licence holder, and is also used in updating licence details or registration.

About Something to look at while the program initialises.

Extracting Text

This program can extract text out of any type of file.

To qualify as text i.e. meaningful words, rather than a collection of visible characters, each potential string of characters has to undergo a variety of tests.

The **INPUT/character set** menu is used to define which characters are to be regarded as text

Those characters which are highlighted in **red** are considered to be valid text characters.

The **INPUT/RECORD FORMATS** menu is used to specify what delimiters might be in use to separate records in the file e.g. word count, 0D0A etc.

Default settings are available for word processing documents and executable programs, but other values can be used (and saved to file for re-use).

To extract text out of a word processing document :-

select **Input / Record formats** on the main menu

select **Settings / Text** on the Record dialog menu

Click on **OK** to accept these settings

Click on **View / Text**

The program will then filter out everything that does not match the Record Format settings.

Please note, this only extracts text - it will not copy any formatting data apart from **tab** positions.

These settings will be retained when you exit the program (unless you change them again).

These defaults will work on most word processing documents.

When using the Copy to Clipboard or Disk commands, the default is to output in **text record** format but this can be changed via the Output/Record formats.

To extract text out of an executable or data file :-

select **Input / Record formats** on the main menu

select **Settings / Program** on the Record dialog menu

Click on **OK** to accept these settings

Click on **View / Text**

A fuller description of how the program can tell words for example, "the cat sat on the mat" or "£1,000" from garbage strings of letters or numbers such as "abk g125" is given later.

Please note, this program was not written as a word processing translator - it has no in-built translation functions to enable this (nor could it cover every word processor even if it did).

To check on the accuracy of the selection, there is an option within the Input/Record Format to display those strings which failed the tests.

The error rate is extremely low - less than 0.5% (even on programs); though there has not been any stringent testing (i.e. none) done on documents containing mathematical equations.

When extracting text - especially from an executable file - there will always be some strings that pass the tests when perhaps they should not.

We have had to accept this because we could not find a way of validating abbreviations, as these vary from country to country.

For example, MCC or BBC are certainly valid if you play cricket or speak english, and should not be discarded even though they are not words.

If **Base 64 decode** is set to **ON** and the file does contain Base 64 encoded data, then the encoded data is decoded before being processed as text.

For a string of characters to be considered as text it has to satisfy the following tests.

Those that can be tailored via the Record Format screen are :-

delimiters	at start		byte or word count - Pascal, Visual Basic programs (**High/Low** and **Low/High** order supported)
	at end	0D0A	most Windows or DOS word processors, text editors
		0A	Unix text processors
		0D	Mac text processors
		00	C type programs
		,	Comma separated variables (CSV)
		other values can be added as required	

break this has the same effect as an end delimiter but the character is considered to be

part of the record and is not discarded.

So, if hex 7E (~) was set as a break character

"Once upon a time~there was ..."   would appear as 2 records

"Once upon a time~"
"there was ..."

length Min ignore text smaller than this value (default 3)

Max any string greater than this value will be split at this length (default 1024, max 6144).

Poss Valid Any string greater or equal to this length will be passed through to the next sequence of tests, regardless of the above tests.

(default value is 12 but very few strings fail the previous tests)

currency This is used in validating currency when used in short strings or column headings

"That costs $20.00"   would be passed as text
"£1,000"   would not - unless £ was set as currency

The default value (£, $, Dm etc.) is taken from your Control Panel International Settings and other currency symbols can be added if needed.

(Date formats, thousand separators and decimal points are also picked up from the same source)

invalid strings as stated earlier this reverses the display so that all the strings that have been rejected as text can be examined. (this does not include any that are less than the minimum length)

In-built tests cover areas such as :-

garbage filter reject nonsense combinations that are often found in executable files, for example   !*, l£, $*

specific tests
hex numbers
decimal numbers
repeat characters
maths symbols
vowel counts
upper/lower/upper case combinations
space counts
punctuation counts

If it passes these tests then the string is **probably** text and displayed as such.

If **Base 64 decode** is set to **Auto on Text** and all the previous tests have failed, then the text is tested to see if it might be an encoded string.

If they are, then any invalid characters in the decoded string are replaced by spaces and the decoded string is re-processed to see if it contains text.

Whilst this is not particularly accurate it does highlight the fact that encoded strings have been encountered and Base 64 Decode can then be switched ON.

(Note - this obviously has a slight impact on performance - you may prefer to switch Base 64 decode off).

When copying to disk, the **OUTPUT** character set and record formats determine what format the file should be written in.

Text record format

The end of each text record is indicated by **hex** 0D0A.

These are the **ANSI** printer commands for line feed and carriage return.

This format is often referred to as ASCII or DOS text and is accepted by virtually every text editor or word processor.

File Identifiers

Many files use the first few characters in the file to act as identifiers to verify what type of file it is.

This is a list of some common identifiers.

Char	Hex	Posn	Extn	File type
BM	424D	0	BMP	Bitmap (Device Independent)
--	B5A2 B0B3 B3B0	0	CAL	Calendar file (calendar.exe)
--	50C3	0	CLP	Clipboard file
--	0000 0200	0	CUR	Icon file describing a Cursor
--	DBA5	0	DOC	Word document
MZ	4D5A	0	DLL	Dynamic Link Library
MZ	4D5A	0	EXE	Executable file
GIF87a	4749 4638 3761	0	GIF	Graphics Interchange Format
PMCC	504D4343	0	GRP	Windows group file (Program Manager)
--	0000 0100	0	ICO	Icon file
--	0108 or 0109	0	IMG	GEM/IMG image file
--	0A	0	PCX	PC Paintbrush
SZ	535A	0	nn_	Compressed file (compress.exe)
II or MM	4949 or 4D4D	0	TIF	Tagged Image File (see **Low/High**)
--	D7CD C69A	0	WMF	Placeable Metafile
--	0109	0	WMF	Memory Metafile
--	0209	0	WMF	Disk Metafile
--	0904	0	XLS/M	Excel spreadsheet (sometimes 0900)

Note - some of these are from documented sources, others are the result of using this program on as many examples as could be found and should not be taken as definitive or even accurate.

PC Paintbrush Z-Soft Corp.

Character definition

A member of the current character set which has a visual representation.

see **ANSI**, **ASCII**, **ASCII+** or **EBCDIC**

Invalid characters are the control characters which have no visual representation and are
unsupported in the Windows environment.

(Note - valid characters are not necessarily the same as **text**.)

Text definition

A string of visible **characters** which satisfy tests to determine if it contains meaningful information - see **extracting TEXT**.

OEM

Original Equipment Manufacturer - the character set installed on the PC by the manufacturer.

ANSI - American National Standards Institute

The following table shows the **ANSI** codes for each **byte** - for example 41 is the letter A.

		Most Significant Byte																		
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**			
	0	†	†	Space		0	@	P	`	p	†	†	Space		°	À	Ð	à		ð
	1	†	†	!	1	A	Q	a	q	†		¡	±	Á	Ñ	á	ñ			
	2	†	†	"	2	B	R	b	r	†		¢	²	Â	Ò	â	ò			
	3	†	†	#	3	C	S	c	s	†	†	£	³	Ã	Ó	ã	ó			
	4	†	†	$	4	D	T	d	t	†	†	¤	´	Ä	Ô	ä	ô			
	5	†	†	%	5	E	U	e	u	†	†	¥	µ	Å	Õ	å	õ			
	6	†	†	&	6	F	V	f	v	†	†	¦	¶	Æ	Ö	æ	ö			
LSB	**7**	†	†	'	7	G	W	g	w	†	†	§	·	Ç	×	ç	÷			
	8	*	†	(	8	H	X	h	x	†	†	¨	¸	È	Ø	è	ø			
	9	*	†	)	9	I	Y	i	y	†	†	©	¹	É	Ù	é	ù			
	A		*	†	*	:	J	Z	j	z	†	†	ª	º	Ê	Ú	ê	ú		
	B	†	†	+	;	K	[	k	{	†	†	«	»	Ë	Û	ë	û			
	C	†	†	,	<	L	\	l	|	†	†	¬	¼	Ì	Ü	ì	ü			
	D		*	†	-	=	M	]	m	}	†	†		½	Í	Ý	í	ý		
	E	†	†	.	>	N	^	n	~	†	†	®	¾	Î	Þ	î	þ			
	F	†	†	/	?	O	_	o	†	†	†	¯	¿	Ï	ß	ï	ÿ			

Values 8, 9, 0A & 0D are backspace, tab, linefeed and carriage return respectively.

† these characters are not supported by the Windows Operating System.

ASCII - American Standard Code for Information Change

ASCII is a subset of the **ANSI** codes - it only covers the **hex** values 00 to 7F as shown below.
(41 is the letter A)

				Most	Significant Byte																
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**		**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**		
	0	†	†	Space		0	@	P	`	p		†	†	Space		°	À	Ð	à	ð
	1	†	†	!	1	A	Q	a	q		†		¡	±	Á	Ñ	á	ñ		
	2	†	†	"	2	B	R	b	r		†		¢	²	Â	Ò	â	ò		
	3	†	†	#	3	C	S	c	s		†	†	£	³	Ã	Ó	ã	ó		
	4	†	†	$	4	D	T	d	t		†	†	¤	´	Ä	Ô	ä	ô		
	5	†	†	%	5	E	U	e	u		†	†	¥	µ	Å	Õ	å	õ		
	6	†	†	&	6	F	V	f	v		†	†	¦	¶	Æ	Ö	æ	ö		
LSB	**7**	†	†	'	7	G	W	g	w		†	†	§	·	Ç	×	ç	÷		
	8	*	†	(	8	H	X	h	x		†	†	¨	¸	È	Ø	è	ø		
	9	*	†	)	9	I	Y	i	y		†	†	©	¹	É	Ù	é	ù		
	A		*	†	*	:	J	Z	j	z		†	†	ª	º	Ê	Ú	ê	ú	
	B	†	†	+	;	K	[	k	{		†	†	«	»	Ë	Û	ë	û		
	C	†	†	,	<	L	\	l				†	†	¬	¼	Ì	Ü	ì	ü		
	D		*	†	-	=	M	]	m	}		†	†		½	Í	Ý	í	ý	
	E	†	†	.	>	N	^	n	~		†	†	®	¾	Î	Þ	î	þ		
	F	†	†	/	?	O	_	o	†		†	†	¯	¿	Ï	ß	ï	ÿ		

Values 8, 9, 0A & OD are backspace, tab, linefeed and carriage return respectively.
† these characters are not supported by the Windows Operating System.

ASCII Plus

Using the Options/Text menu it is possible to extend the normal **ASCII** character set to include some of the **ANSI** character set e.g. £, ©, ® without having to opt for the full ANSI version.

EBCDIC - Extended Binary Coded Digit Interchange Codes

EBCDIC coding is rarely used on PCs; they are more common on mainframes but some word processors use them (DCA/RFT format).

The following table shows the **EBCDIC** codes for each **byte** - for example C1 is the letter A.

Most Significant Byte

		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**	
	0	†	†	†	†	Space		&	-	º	Ã	Ê	Ñ	Ø	{	}	\	0
	1	†	†	†	†	NBSP		©	/	»	a	j	¯	Ù	A	J	‹	1
	2	†	†	†	†	¡	ª	²	¼	b	k	s	Ú	B	K	S	2	
	3	†	†	†	†	¢	«	³	½	c	l	t	Û	C	L	T	3	
	4	†	†	†	†	#	¬	´	¾	d	m	u	Ü	D	M	U	4	
	5	*	†	*	†	¤		µ	¿	e	n	v	Ý	E	N	V	5	
	6	†	*	†	†	¥	®	¶	À	f	o	w	Þ	F	O	W	6	
LSB	**7**	†	†	†	†	¦	~	·	Á	g	p	x	ß	G	P	X	7	
	8	†	†	†	†	§	°	¸	Â	h	q	y	à	H	Q	Y	8	
	9	†	†	†	†	¨	±	¹	`	i	r	z	á	I	R	Z	9	
	A		†	†	†	†	[	]	|	:	Ä	Ë	Ò	â	è	î	ô	ú
	B	†	†	†	†	.	$	,	£	Å	Ì	Ó	ã	é	ï	õ	û	
	C	†	†	†	†	<	*	%	@	Æ	Í	Ô	ä	ê	ð	ö	ü	
	D		*	†	†	†	(	)	_	'	Ç	Î	Õ	å	ë	ñ	÷	ý
	E	†	†	†	†	+	;	>	=	È	Ï	Ö	æ	ì	ò	ø	þ	
	F	†	†	†	†	!	^	?	"	É	Ð	×	ç	í	ó	ù	ÿ	

Values 05, 0D, 16 & 25 are tab, carriage return, backspace and linefeed return respectively.

† these characters are not supported by the Windows Operating System.

Note - all the characters are visually identical to those used by **ANSI** and **ASCII**. It is only their **hex** values that are different.

The program also supports both ISO and US versions of EBCDIC.

The table above shows the ISO (West European) version, the only differences in the US version are hex 57, 7B and A1 contain ¯, **#** and **~** (overscore, hash and tilde) respectively.

Character sets and translation tables

When a file is produced on one computer, and is to be displayed correctly on a different machine then either **both** must use the same character set or the file has to be **translated**.

In other words both computers must recognise which hex values represent specific letters, numbers, punctuation etcetera.

Most PC's have adopted the **ANSI** character set for exchanging data so they all can read the same files - but not all machines use ANSI.

For example, a pound sign **£** is represented by the hex values **A3, 7B** or **9C** when using the **ANSI**, **EBCDIC** or **OEM** character sets respectively - but is not defined in the ASCII set.

MicroFile recognises three international character sets **ANSI**, **ASCII**and **EBCDIC**.

The ASCII definition is identical to the ANSI set but only covers the range 00 to 7F.

This program allows for an extended version of ASCII, which we have called **ASCII+**.

With this it is possible to extend the standard ASCII character set with some of the ANSI definitions, for example **£**, **©**, **®**, **§** - so that these are also regarded as part of the character set without opting for the full ANSI version.

To select/deselect a character as text, use **INPUT/CHARACTER SET** and click on whichever character is **required**/**not required** (see **extracting text**).

Most values in the ranges 00 to 2F and 7F to 9F can also be translated to any other value that may be required.

For example, to convert a **tab** delimited text file to a **comma** separated variable format :-

click on hex **09** (ANSI tab)
a dialogue box will appear
enter a **,** in the character box (or 2C in the hex box)

All tab characters will then be translated to commas and an output file could be created using **Copy_to_Disk**.

There are also options to select the OEM character set or to define any other translation table.

However, with the **OEM** set not all ANSI characters are represented.

For example, **È**, **Ê**, **Ë** exist in ANSI but not in the OEM U.S. standard set (DOS page 437).

A new translation table would have to be created to translate an ANSI file to OEM where those particular characters would probably be converted to **E**.

Example convert an ANSI text file to all upper case characters

select **INPUT/CHARACTER SET**

select **User defined**

select **File/New**

select **Base on ANSI**

click on **a** and change the value to **A**

repeat for **bcdefghijklmnopqrstuvwxyz**

select **Done**

select **File/Save As** and save the file as **ANSI2UC**.

select **OK**

All lower case characters will now be translated to upper case, and the file **ANSI2UC** is then available for any future use.

Tab positions

A pre-determined position within text, generally used to align columns within a document.

Most word processors or text editors default to every 1/2 inch or every 8 characters.

The tab positions can usually be altered to suit the layout required.

The hex code for a tab is 09 in **ANSI** and **ASCII**, 05 in **EBCDIC**.

Delimiters

A **byte** (or character) used to indicate the start and/or end of data.

Usually a value that does not correspond to a keyboard character(s) such as

0D0A	in text files to enable carriage return & line feed
00	in compiled programs to indicate end of strings
¶	in visual displays such as word processors

Byte

A sequence of 8 bits of binary data.

It can hold the values 0000 0000 through 1111 1111 in **binary**

which is the equivalent of 00 to FF in **hexadecimal**

or 0 to 255 in decimal

Binary

A number system using **base 2**; characters **0** and **1** are used to represent each value.

decimal	**binary**
0	0
1	1
2	10
3	11
4	100
5	101

Because computers record data in **bytes** (8 bits) binary numbers are generally written
in groups of 4 with leading zeroes to make them more readable e.g.

0001 0001   1101 0111

Hexadecimal

A number system using **base 16**; characters **0123456789ABCDEF** are used for each value.

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
10000	10	16

Because computers record data in bytes and each **byte** requires 2 hexadecimal numbers,
they are generally written in groups of 4 to make them more readable e.g.

FFFF 0A11   ABCD 1234

Definitions

The following definitions are used in this program

Byte	8 bits
Word	2 bytes
Long	4 bytes
Double	8 bytes

Exit

Exits the program

(Note all current settings are saved at this point)

Title Bar

Displays the program name and the current file in use

File Menu

Provides the following options :-

New	Opens a new file
Open	Re-opens a closed file
Close	Closes an open file
Print	Copies the current screen details to the default printer
Exit	Exits the program

Edit Menu

Provides the following options :-

Copy to Clipboard		Copies screen details to the clipboard
Copy to Disk		
	Normal	Copies the file to disk in the current screen format
	Decode Base 64 files	Extracts & decodes any Base 64 file & copies to disk
	Encode UU format	Copies to disk encoding to UU format
	Encode MIME format	Copies to disk encoding to MIME format
Goto		Go to a file position
Find		Find hexadecimal or character string
Find Next		Find next occurrence of string
Find Previous		Find previous occurrence of string
Replace		not implemented yet - see **future developments**
Edit mode		not implemented yet - see **future developments**

View Menu

Provides the following options :-

Address	Switches address on or off
Decimal	Display file position in decimal
Hex	File position in hexadecimal
Base 0	First byte in file is address 0
Base 1	First byte is address 1
Columns	Vary the number of characters displayed (or scroll sideways)
Font	Switch between the various fonts and sizes
Hex & Char	Display **hex** then **character** representation (left to right)
Char & Hex	Display characters above hex values
Hex only	Hex only
Char only	Character only
Text - records	Display **extracted text** records from file
Text - word wrap	Display **extracted text** records from file in word wrap mode
Base 64 decode	Selects Base 64 options
On	Searches for any Base 64 encoding and displays decoded values
Off	No search, display actual file values
Auto on text	No search; any text failing tests tested to see if Base 64 encoded

Options Menu

Provides the following options :-

Background	Change the background colour of the main screen
Foreground	Change the foreground (text) colour
As Above	Display "nn lines as above" instead repeating line after line
Screen de-limiters	Visible markers to show start and end of text; default **+......+**
Unprintable chars	Change the visual presentation of unsupported characters
Calculator	Hex translations - see **calculator**
Base date	Set base date i.e. day 0 for date calculations

Input File Menu

Provides the following options :-

Character Set	Choose between **ANSI**, **ASCII**, **EBCDIC** or **ASCII+**, **OEM** or user defined tables
Record formats	Change the settings that define the **extracted text** requirements

Output File Menu

Provides the following options :-

Character Set	Choose between **ANSI**, **ASCII**, **EBCDIC** or **ASCII+**, **OEM** or user defined tables
Record formats	Change the settings that define the **extracted text** requirements

Help Menu

Provides the following options :-

Contents	The help Contents page
Tooltips	Toggles tooltips ON/OFF
Licence Details	Used to enter or print out licence details
About	About this program

Exit

Exits the program

(Note all current settings are saved at this point)

New File

Opens a new file

File Close/Re-open

 closes the current file

 re-opens the file

Address on/off

Switches the address display on or off.

Address style

 change address to **hexadecimal**

 change address to decimal

Address base

 set first **byte** in file to address 0

 set first byte to address 1

Scroll bar

Normal mode - varies the number of characters displayed on the screen
(click elsewhere on the screen to exit this mode - or use Esc)

Text records - scrolls left and right along the records
(click on the button a second time hides the scroll bar - or use Esc)

Fonts

Select which fixed pitch font should be used.

Goto

Go to a new file position

This can be specified in **hexadecimal** or decimal

It can be absolute i.e. go to postn 23 or relative +10, -4

(+ or - act as keys for this function)

Search

Searches the file for a string.

The string can be in **hexadecimal** or text.

Find next, previous, from start or end are available as well
as case sensitive and not equal to.

Screen prints

Copies the contents of the current screen to the default printer

Copy to clipboard

In normal mode it copies the contents of the current screen to the clipboard.

In text mode it copies every record that is visible on screen to the clipboard.

(Note - the whole record is copied even if only part is displayed on the screen)

Copy to disk

This copies the file to disk in the format specified via the Output and View
menus.

Calculator

This performs various **hex** translations.

It can convert signed/unsigned **integers**, **floating point** and **currency** formats, date and time, file pointers and bit patterns.

For full details see **calculator**.

Help Panel

This is empty when no file has been selected, otherwise it displays the full hierarchic name, size and date last amended of the current file.

Placing the cursor over any control and holding a mouse button down will display a hint in this panel as to what that command does.

Releasing the mouse while still over the control will action the command.

When displaying a compressed file both compressed and uncompressed sizes are shown.

For example, a compressed file of 1024 bytes which unpacked to 3172
would be shown

c:\fred 1024 (3172) when viewed in it's actual state and
c:\fred 3172 (1024) if viewed in it's expanded form

Unsupported characters

There are several **hex** values which are "not supported" in Windows.

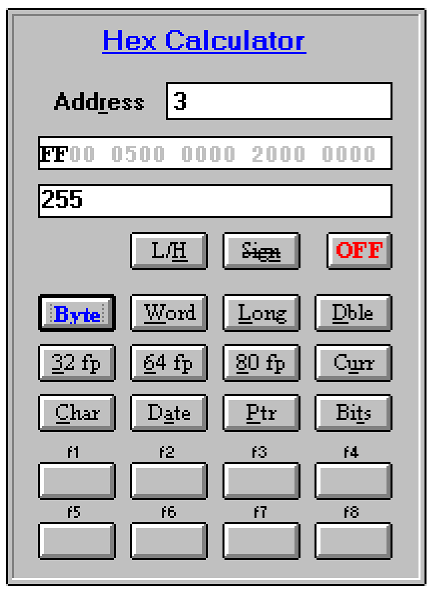
They are used as commands to drive hardware such as printers or modems and have no visual representation.

See **ANSI**, **ASCII** or **EBCDIC** for further details.

Calculator

The calculator can started by clicking on it's icon  or via the Edit menu

Click the part that you want to know more about.



The mouse buttons allow you to drag the calculator around on the screen.

However, the RIGHT mouse button also REFRESHES the parts the other buttons don't reach, by aligning the screen and calculator addresses.

with the mousepointer	over the calculator	updates the screen address
	over the screen	updates the calculator address

If a particular button on the calculator is shown in grey rather than black, it means that either that option is not available or it is out of range.

For example, the value FFFF (65535) could not be a pointer to an absolute position in a file, if the file was only 20,000 bytes long.

So if Ptr and Abs were selected only the byte option would be shown in black.

Other functions are

Low/High or **High/Low** byte order

signed / **unsigned** arithmetic

floating point arithmetic

date functions

file pointer functions

bit patterns

Note: the only calculator function affected by the choice of character sets is the character display.

The hexadecimal values are **never** translated by character set.

Signed and Unsigned arithmetic

With unsigned arithmetic (~~Sign~~) then the entire range is considered positive

i.e a byte contains 0 - 255 (00 to FF)

With signed arithmetic (Sign) then FF = -1, FE = -2 etc

so the range becomes -128 to 127 (80 to 7F)

The ranges that can be held in byte, word, long and doubles are :-

B	byte	8 bits	unsigned	0 to 255
			signed	-128 to 127
W	word	2 bytes	unsigned	0 to 65535
			signed	-32768 to 32767
L	long	4 bytes	unsigned	0 to 4,294,967,295
			signed	-2,147,483,648 to 2,147,483,647
D	double	8 bytes	unsigned	0 to 18,446,744,073,709,551,615
			signed	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

(Note a currency variable is a signed double divided by 10,000 which gives the range -922,337,203,685,477.5808 to 922,337,203,685,477.5807)

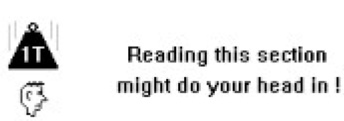
Method of calculation

Visual Basic Version 3 provides two integer formats, INT and LONG - both are signed word and long words respectively.

This program calculates all values outside the scope of these formats by using an integer array to hold intermediate results during long multiplication and eventually converting this into a string.

Negative numbers are calculated using 2's complement and adding a minus sign to the string.

Accuracy on integer calculation is 100%



In unsigned arithmetic all the values are assumed to be positive and adding two bytes together is not a problem if the result can still be stored in a single byte e.g. 01 + 05 = 06.

However, if the result is too big to be stored in one byte then the result has to be carried over to be stored in a second byte for example adding FF to FF

```
        FF          1111  1111  +
        FF          1111  1111
                    ----------------
                        1 1111  1110       which is 01 FE
```

(Note that as far as this byte was concerned the result was FE)

Subtraction can be done just as easily as long as the result is a positive number.

However, there is no position in a byte to hold a minus sign - so a method to represent negative numbers had to be devised.

All computers contain registers to hold numbers, and flags to indicate results of an operation.

These flags are usually zero, negative and carry (amongst others).

Consider a program loop in machine code

```
              LDA     3          load register A with the value 3

    LOP         .                perform various other commands
                .
              SUBA    1          subtract 1 from register A
              BPL     LOOPif result positive then branch back to LOOP
```

This would be performed until the value in register A became negative.

What does "negative" mean and how does signed arithmetic work?

In most branches of mathematics a positive value of some amount <u>added</u> to a negative value of the same amount should give zero.

In signed arithmetic, the convention is if the highest bit in the number is 0 then the number is positive, if it is a 1 then the number is negative.

This method - more accurately known as 2's complement arithmetic - is used by virtually every modern computer.

In effect, the 2's complement of a number is another number that, when they are added together, gives zero.

In practice, this means that the complement of

```
    01 is FF     0000 0001   +   1111 1111   =   0000 0000
    02 is FE     0000 0010   +   1111 1110   =   0000 0000
    03 is FD     0000 0011   +   1111 1101   =   0000 0000
```

when the result of the addition is carried forward to the next pair of bytes.

It is known as two's complement because one's complement got there first (see later).

To form a 2's complement of a number, you invert all the bits and add 1.

For example, the complement of 3 is calculated by

decimal 3 in binary is	0000 0011	
inverting all the bits gives	1111 1100	
adding 1	0000 0001	

	1111 1101	which corresponds to hex FD.

Using 2's complement has several advantages.

(1) a byte, word, long word etc. can hold negative as well as positive numbers

(2) many machine instructions only require **relative** values as in Branch if Plus, Minus, Equal, Not Equal etcetera.

(3) only code to add numbers is needed - subtraction is performed by forming the complement and adding the numbers.

On early computers, one's complement arithmetic was used.

In this method, the complement was formed only by inverting the bits.

This had the effect that a byte could hold values in the range -127 to +127 (hex 80 and 7F) but with the concept of a plus **and** minus zero (00 and FF) !

Floating Point arithmetic

The calculator can convert 32, 64 and 80 bit floating point numbers to decimal, in the ranges

32 bit single 4 bytes -3.4 x 10E-38 to 3.4 x 10E38 precision 7 digit

64 bit double 8 bytes -1.7 x 10E-308 to 1.7 x 10E38 precision 15 digit

80 bit long 10 bytes -3.4 x 10E-4932 to 1.1 x 10E4932 precision 15 digit

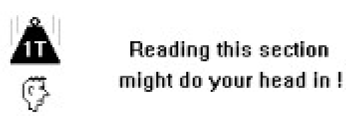
Note an 80 bit floating point actually gives 19 digit precision but Visual Basic will only work to 15 (we had to use various mathematical chicanery to get that far).

To test the accuracy of our results, we generated a file filled with random numbers in the range 0 to 255 i.e. bytes with random values in them.

We then used VB and the calculator to intepret these as if they were 64 bit floating point numbers.

Comparing the results, showed an erratic error rate in these interpretations.

As a consequence, and as our calculations are also limited by VB, we feel that the method adopted is accurate to 13 digits, and probably +/- 2 in the 14th digit.



The format for holding a floating point number is basically the same for all three versions.

Each sequence of 32, 64 or 80 bits hold the sign, the exponent and the significand.

To write the decimal number 124.8 in this type of format then

the sign is 0 as it is a positive number

124.8 can be written in exponential terms as

1.248 E2 where E2 means 10 to the power 2 (i.e. x 100)

a number written in this format, where there is only 1 (non zero) digit in front of the decimal point is known as a normalised number

This same number could then be written in sign, exponent and significand order as

0-2-1248

Note that the decimal point is not recorded - it's position is implied because the number has been normalised.

In using this method to hold **binary** floating point numbers, no separators (-) are used, but the sign, exponent and significand are each allocated a fixed number of bits.

If this was all there was to floating point formats, it would not be too bad but life is never that simple.

For a start, in a normalised binary number, the leading digit of the significand will always be 1.

As this integer bit's value is known, it does not have to be stored and consequently the format can be used to give one more bit of accuracy.

Consequently, although the significand of a 32 bit number is only allocated 23 bits of storage (see the table below) the implied bit increases this to 24 bits of data.

However, this implied method is only used on 32 and 64 bit numbers but not on 80 bit numbers.

The bits used by each method are

Sign	1	1	1
exponent	8	11	14
integer bit of significand	implied	implied	1
significand	23	52	64
	---	---	---
	32	64	80

Also, a method has to be used to indicate whether the exponent is positive or negative and in this instance a biased exponent is used.

Unlike signed arithmetic which is based on plus or minus from zero, a different value is chosen to **represent** the zero position.

The actual bias used for 32, 64 and 80 bit representation is 127, 1023 and 16383 respectively.

While this appears/is complicated, in practice it simply requires the bias to be subtracted from the calculated value of the exponent.

The following example illustrates how this program calculates the value of a floating point number.

Consider 4 bytes containing **42F6 E9D5** in **High/Low** order as a 32 bit floating point number.

writing this as a bit pattern gives

4	**2**	**F**	**6**	**E**	**9**	**D**	**5**

0100 0010 1111 0110 1110 1001 1101 0101

splitting this into the sign, exponent and significand components gives

1	8	23
0	**1000 0101**	**111 0110 1110 1001 1101 0101**

The sign bit is zero, so this is a positive number.

The exponent is **1000 0101** or **85** hex, which is **133** in decimal.

Subtracting the bias, the exponent value is **133 - 127 = 6**

The significand has an **implied** leading bit, so the actual value is

	1111	**0110**	**1110**	**1001**	**1101**	**0101**
or	**F**	**6**	**E**	**9**	**D**	**5**

F6E9D5 in hex gives **16181717** in decimal.

However, this value assumes that the significand is an integer - it does not yet reflect the fact that it contains a **binary** point (in this instance 23 positions in from the right).

Calculating the log of this value to base 2 gives

LOG(16181717) / LOG(2) = 23.9478613607286

The program then normalises the value by subtracting the binary position to give

23.9478613607286 - 23 = 0.9478613607286

Adding in the exponent value gives

6 + 0.9478613607286 = 6.9478613607286

And finally, taking the anti-log

EXP(6.9478613607286 x LOG(2)) = 123.456703186035

As the accuracy of this number is only to 7 digits, the program truncates the number to 8 digits and rounds up, discards the last digit and removes any trailing spaces.

It then converts this to a string and, if it was a negative number, adds a minus sign at the start and the result is then displayed on the calculator.

42F6 CCCD => **123.4567**

However, there are further complications when dealing with 80 bit floating point numbers.

The maximum value of the exponent can be as high as 4932.

Calculating the exponent is not a problem, but when added to the LOG value this could result in requiring an anti-log of say,   4932.123456

Visual Basic can only cope with EXPonential values up to 709.782712893

The program attempts to cope with this situation by dividing the LOG value by 2   (i.e. taking the square root) until it is less than 700.

It then converts the number to decimal, and normalises it.

Then, treating the decimal exponent and significand as separate entities, it squares each of them the same number of times that it took square roots.

It finally converts these calculations to a string and then displays the result.

Also, 80 bit floating point numbers have a generally accepted accuracy of 19 decimal digits, but the maximum VB can cope with is 15.

As we do not know of a fast way of emulating Logarithm and   Exponential functions in Visual Basic, we have settled for this limitation.

If anyone does know of any algorithm to achieve both speed and accuracy, could they let us know, and we will try to incorporate it into the next release.

UU coding

When transmitting binary data over networks or via a modem, it is quite possible that the data may contain a sequence of bytes that the transfer protocol could interpret as control information.

When this happens it often results in aborted transfers.

Various method have been arrived at which encode each data byte into **character** values so they cannot be mis-interpreted as control sequences.

Base 64 encoding works by taking groups of 3 bytes - or 24 bits and splits these into 4 * 6 bits and outputing these as 4 character bytes.

UU encoding would encode the letters ABC as follows :-

```
ABC in hex is           4142 43

                        4    1     4    2     4    3
or in binary            0100 0001  0100 0010  0100 0011
                        |||||||--------  ||||||||-------

these are split into    0001 0000  0001 0100  0000 1001  0000 0011
4 groups of 6 bits

hex 20 is added to these   0010 0000  0010 0000  0010 0000  0010 0000
(space character)       ---------  ---------  ---------  ---------
                        0011 0000  0011 0100  0010 1001  0010 0011
                        3    0     3    4     2    9     2    3

or, in hex              3034 2923

or, in ansi characters  04)#
```

Note - there is a slight problem with hex 00.

Using this method hex 00 is translated into a space character (hex 32).

With some network protocols, trailing spaces in a line of text are truncated to reduce the volume of data that has to be transmitted.

To avoid this problem the UU format also accepts hex 60 (the back-quote character `) to represent hex 00, as this prevents any truncation occurring.

The output **text** file uses the following format :-

begin nnn orig_file_name
encoded lines
.
.
end

where **nnn** is the file access **mode** (only applies to UNIX files).

The first character of each encoded line is equal to the number of valid characters in the string (plus hex 20) followed by a maximum of 45 bytes from the original file.

Most ot the text records in the output file result in line lengths of 1 + 60 and UU encoded files can be recognised with a text editor as virtually every line starts with **M....** followed by 60 characters.

Access mode (UNIX files)

The UNIX filing system is usually run on a shared machine and so has more file security features than DOS.

It is possible to set separate read, write and execute permissions for file owner, group of owners or globally for all users.

This program program sets file access mode to **600** which gives

```
                          R W E
owner  = 6 or in binary   1 1 0
group  = 0                0 0 0
others = 0                0 0 0
```

so only the file owner is given read and write permissions.

As the DOS/Windows operating systems do not have the user/password concepts these settings are ignored when decoding a UU file.

MIME coding

When transmitting binary data over networks or via a modem, it is quite possible that the data may contain a sequence of bytes that the transfer protocol could interpret as control information.

When this happens it often results in aborted transfers.

Various method have been arrived at which encode each data byte into **character** values so they cannot be mis-interpreted as control sequences.

Base 64 encoding works by taking groups of 3 bytes - or 24 bits and splits these into 4 * 6 bits and outputing these as 4 character bytes.

MIME encoding would encode the letters ABC as follows :-

```
ABC in hex is              4142 43

                           4    1     4    2     4    3
or in binary               0100 0001  0100 0010  0100 0011
                           |||||||--------  ||||||||-------

these are split into       0001 0000  0001 0100  0000 1001  0000 0011
4 groups of 6 bits

each of these is then processed as follows (values in decimal)

                           Select  Case n
                                   Case 62
                                         n = 43
                                   Case 63
                                         n = 47
                                   Case 52 To 68
                                         n = n - 4
                                   Case Is >= 26
                                         n = n + 71
                                   Case Else
                                         n = n + 65
                           End Select

as each of these falls     0100 0001  0100 0001  0100 0001  0100 0001
into the CASE ELSE         ---------  ---------  ---------  ---------
(dec 65 = hex 41)          0101 0001  0101 0101  0100 1001  0100 0011
                           5    1     5    5     4    A     4    4

or, in hex                 5155 4A44

or, in ansi characters     QUJD
```

The output **text** file uses the following format :-

MIME_Version: 1.0
Content-Type: APPLICATION/octet-stream; name="test1.txt"

Content-Transfer-Encoding: BASE64
Content-Description:
encoded lines
.
.
blank line(s) or end- of-file indicate end of data

Unlike UU code, MIME encoding does not specify the number of characters in each encoded line.

Each line is **always** a multiple of 4 characters and usually a maximum of 60 bytes long.

If the length of the original file is not a multiple of 3 characters, then the final encoded line is padded out with equal (=) signs, which are ignored on decoding.

Address

Displays the current position in the file (in hex or decimal; base 0 or 1)

The value can be changed by over typing, using **+** or **-** or via the mouse.

Clicking with the RIGHT mouse button :-

over the calculator updates the screen with the calculator address
over the screen updates the calculator with the screen address

File contents

Displays the file contents in **hexadecimal** at the current address.

The bytes in focus are displayed in **black**, while those not in focus shown in **grey**.

For example, when converting a **word** to decimal, the file contents would be shown

1234 5678 9ABC DEF0

Display line

Displays the result of the current conversion.

Low/High order

Most of the world write numbers in High/Low order - PC processors do not.

For example, one thousand two hundred and thirty four is written as 1234 - high/low order

A hex number 1234 would be stored as 34 12 by a x86 processor.

However, applications that write directly to file are still more likely to use high/low format.

This button toggles between the two views and is effective on all other calculator functions.

Note - Tagged Image Files (TIF) can be in either format. The first two bytes in the file contain either II or MM to indicate Intel or Motorola format, L/H and H/L order respectively.

Signed arithmetic

Switches between **signed** (**Sign**) and **unsigned** (~~**Sign**~~) arithmetic.

Applies to all **integer** arithmetic and pointer calculations.

Off

Switches the calculator off.

Clicking on the screen with the LEFT mouse button or using the **ESC** key has the same effect.

Byte conversion

Converts a **byte** to a decimal **integer**

Word conversion

Converts a **word** to a decimal **integer**

Long word conversion

Converts a **long word** to a decimal **integer**

Double long word conversion

Converts a **Double long word** to a decimal **integer**

32 fp

Converts a 32 bit **floating point** number to decimal

64 fp

Converts a 64 bit **floating point** number to decimal

80 fp

Converts an 80 bit **floating point** number to decimal

Currency

Converts a **currency** number to decimal

Integer variable

An integer is a whole number with no fractional parts

such as 1, 15, 123, -3

Floating point variable

A floating point number is a real number which has fractional parts

such as 1.0, 1.25, 0.123, -12.34

Currency variable

A currency variable is a signed **double** integer with a fixed decimal offset.

Effectively, it is a signed double integer divided by 10,000 .

Char

This displays the file contents in the current character format.

Date

This provides various date and time formats available using the **function keys**.

Date

This provides file pointer interpretations using the **function keys**.

Bit patterns

This provides bit level displays using the **function keys**.

Function keys

The function keys provide different facilities depending on which main key has been selected.

See **Functions**

Byte, word long or double

The hex values shown in **black** highlight the bytes under examination.

In this example, a word is being examined, so 2 bytes are displayed in black

Byte in focus

This highlights which **byte** is being examined.

Byte, word long or double

These hex values are shown in grey to differentiate them from the bytes under examination.

Bit position

This displays the left-most bit position of the byte currently being examined.

All bit positions are calculated from 0.

Binary display

This displays the bit pattern of the byte currently in focus.

Bits command

This is highlighted to show that a binary display has been selected.

Command buttons

black - the option is available.

blue - which option has been selected
grey - the option is not available

Arrow buttons

These are used to select which byte is currently in focus.

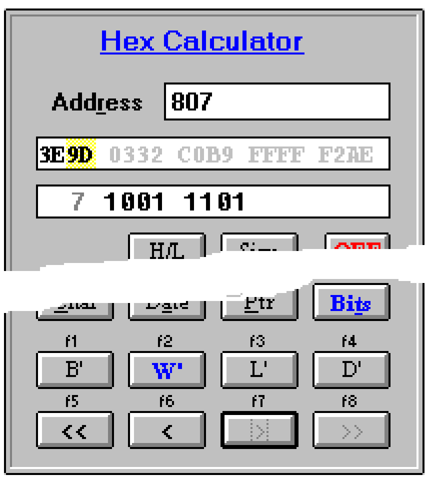
The left arrows move to the higher bytes, right arrows to the lower bytes (single arrows move one byte at a time, double arrows jump to highest and lowest)

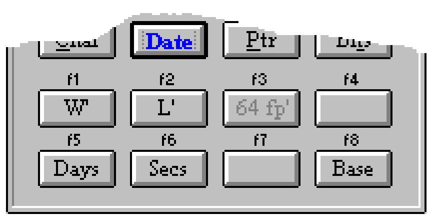
Note - when viewing in **High/Low** order, the display moves in the direction of the buttons, but reverses when examining Low/High order, and yes, it is confusing !

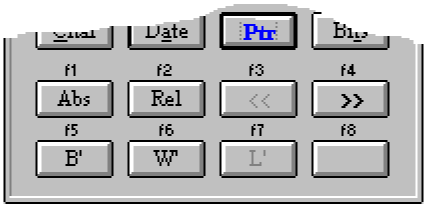
Calculator function keys

These provide different facilities depending on the main key that has been selected

Click on the part you wish to know more about.







Date
Ptr
f1
Abs
f2
Rel
f3
<<
f4
>>
f5
B'
f6
W'
f7
L'
f8

Pointer command

This command provides file pointer functions.

Pointer Absolute or Relative address

This selects whether the pointer is based from the start of the file or relative to the current position.

Return Pointer

This command will be inactive until a jump to a pointer position has been performed.

This button will then return you to the point you jumped from.

Jump Pointer

This button allows you to navigate the file by jumping to the file position that shown in the display window.

It will be greyed out if the position pointed at lies outside the file or if it is pointing at the current calculator position.

Date command

This command provides date interpretations.

Date/Word

The word value is used in calculating the
date/time.

Date/Long

The long word value is used in calculating the
date/time.

Date/Word

The date corresponding to number of days since base date using 64 bit floating point representation where

the integer part is days since base date

the fraction is time since midnight

Date/Days

The date corresponding to number of days since the base date.

Date/Secs

The time corresponding to minutes since
midnight.

Date/Base

Select which base date should be used.

The base date screen is displayed, showing the more common date conventions plus there is an option to select any base date in the range 0-9999 B.C.

Future developments

The proposed requirements for the edit facility are still being drafted (any input would be welcome) and it is expected to have this facility available in Q4 1996.

Note these are partly specification and partly wish list - the next version of MicroFile may contain all, some, or none of these, depending on time constraints and any feedback we receive.

Remember - there is no additional payment required from Registered users to use a new version.

Your password will be valid on all updates for the duration of your licence period.

This program is distributed as shareware and we expect you have received this copy via friends or bulletin boards and any new versions will be released using the same method.

If you wished us to mail you a new version, then the only cost you would incur would be a nominal charge (from our point of view) to cover copying and shipping expenses.

Edit format

No edits will be allowed on compressed/encoded files or those with read only, system or hidden attributes set.

No changes will be made direct to the file; all edits will be buffered (on disk if required) until the file is closed.

There will then be the option to apply these immediately, at a later time or to discard them.

There will be options to

(1) apply the edits to the original file
(2) rename the original, merge original and edits into a new file with the original name
(3) merge original and edits into a new file of a different name

Full, unlimited undo facility - selectively as well as last change.

Full roll back and forward capability to cope with any interruptions, power failures etc. while the changes are being applied.

If the edits are being applied at a later time, full checking of file properties (date/time changed, size) and that the file edits are still valid i.e. the existing file still has it's original values at the required change positions.

Editing will be by over typing either in hex or character in any screen display or via the calculator.

The calculator would be used to type in decimal values and have them converted to their respective formats.

For example, a field containing the decimal value 123.4 as a 32 bit floating point number in Low/High format would actually contain hex CDCC F642.

If the actual value should have been 12.34 then the correct hex values would be A470 4541 - which is a little bit difficult to calculate in your head.

Security

Individual password control.

Separate read and write permissions at Global, Group and Individual levels as well as drive, directory and file levels.

These would apply to remote as well as local systems.

Restricted access will be both positive and negative i.e. "all but" and/or "none but" .

Calendar

At present, all date calculations are done using the Gregorian calendar.

If there were sufficient demand, then other calendars could be added e.g. Julian, Solar, Islamic, Japanese etc. (but you might have to tell us how the calendar is calculated !).

Please let us know if you have any other requirements, problems in using this program or suggestions to improve it.

The people we have embarrassed into testing this program for us have all tended to use it in areas that we never considered when first developing it.

Their feedback has been extremely useful (frustrating and ******* annoying at times) which has resulted in, we think, a much better utility than it would have otherwise been.

We would like to thank them for their help in this and hopefully, future developments and appreciate very much that they still talk to us - even if the language cannot be repeated here.