XL/XE ONLY

Original documentation translated and compiled by Western New York Atari Users Group

TURBO BASIC COMMAND LIST

Turbo Basic is fully compatible with Atari Basic and can be used to run any Atari Basic program at faster speed. Simply boot up the Turbo Basic disk and proceed as usual. The only restriction is that you will need to re-boot after using DOS (although there are inbuilt DOS functions in Turbo Basic).

There are many extra features in Turbo Basic however and the complete list follows. Programs may be typed in upper or lower case. The language itself actually takes up less space than Atari Basic!

DISK I/O

BLOAD	BLOAD *"D:name"*	Binary loads file name (DOS option L with /N).
BRUN	BRUN *"D:name"*	Binary load and run file name (DOS option L).
DELETE	DELETE *"D:name"*	Deletes the file name (DOS option D).
DIR	DIR	Disk directory (DOS option A).
	DIR *"Dn:*.*"*	Directory of drive n, note that wildcard extenders may be used.
LOCK	LOCK *"D:name"*	Locks the file name (DOS option F).
RENAME	RENAME *"D:old,new"*	Renames the file name (DOS option E).
UNLOCK	UNLOCK *"D:name"*	Unlocks the file name (DOS option G).

GRAPHICS

CIRCLE	CIRCLE *x,y,r*	Plots a circle with center at x,y and radius r.
	CIRCLE *x,y,r,r2*	R2 is an optional 'vertical radius' for true circles or ellipses.
CLS	CLS	Clears the screen.
	CLS *#6*	Clear screen opened in channel 6.
FCOLOR	FCOLOR *n*	Determines fill color.
FILLTO	FILLTO *x,y*	A fill command analagous to the BASIC commands POSITION x,y: XIO 18,#6,0,0,"S:"
PAINT	PAINT *x,y*	Another type of fill command, this one is a recursive routine that will fill any closed object as long as x,y are inside it.
TEXT	TEXT *x,y,a$*	bit-blocks text in a$ at x,y.

MEMORY

DPOKE	DPOKE *m,v*	Pokes location m,m+1 with 2-byte integer v (0 < = v < = 65535).
MOVE	MOVE *m,m1,m2*	Block transfer; moves m2 (number of bytes) from starting position m to new sta:ting position m1.
-MOVE	-MOVE *m,m1,m2*	Same as MOVE but copies starting with the last byte of the block.
BPUT	BPUT *#n,adr,len*	Block Put; same as FOR I=0 TO len-1:PUT #n,PEEK (adr+I):NEXT I
BGET	BGET *#n,adr,len*	**Block Get; same as FOR I=0 TO len-1:GET #N,A: POKE adr+I):NEXT I**
%PUT	%PUT *#n,a*	Until now, there was no convenient way to put numeric values onto disk or cassette files other than by using PRINT, which converted them to strings first, a slow and cumbersome process. %PUT puts the number to the device 'as is,' in 6-byte FP format.
%GET	%GET *#n,A*	Get a number stored with %PUT from the device and store it in variable A. Again, this is much faster than using 'INPUT #n, A'.

STRUCTURED PROGRAMMING

REPEAT	REPEAT	Start a REPEAT-UNTIL loop.
UNTIL	UNTIL *<c>*	Terminate when condition <c> met.

WHILE	WHILE <*c*>	Start a WHILE-WEND loop to end when condition <c> met.
WEND	WEND	Terminate a WHILE-END loop.
ELSE	ELSE	Optional extension for IF. The IF condition must not be followed by a 'THEN', but terminated by end-of-line or colon.
ENDIF	ENDIF	Ends an IF-ELSE-ENDIF or IF-ELSE condition. Note that this allows an IF condition to span more than one BASIC line, provided the 'IF' statement is structured as shown in Note 4.
DO	DO	Starts an 'infinite' DO loop.
LOOP	LOOP	Cycle back to the start of a DO loop.
EXIT	EXIT	Exit a DO-LOOP loop.
PROC	PROC *name*	Start definition of procedure.
ENDPROC	ENDPROC	End definition of procedure.
EXEC	EXEC *name*	Execute procedure name.

GENERAL PROGRAMMING

PAUSE	PAUSE *n*	Pause processing for n/50 seconds.
RENUM	RENUM *n,i,j*	Renumber the program starting at line n,first number is i, increment is j. This function will handle GOTOs, TRAPs, and all other line references except those which involve variables or computed values.
DEL	DEL *n,i*	Delete lines n-i.
DUMP	DUMP	Display all variables and values. For numeric arrays, the numbers are the DIMed values plus one. For strings, the first number is the current LENgth of it and the second number is the DIMed size of it. DUMP also lists procedure names and labels with their line values.
	DUMP *name*	DUMP to device name, such as "P:" or "D:DUMP.DAT".
TRACE	TRACE	Trace program during execution.
	TRACE -	Turns trace mode off (Default).
DSOUND	DSOUND *n,f,d,v*	Form of SOUND which activates channel-pairing for increased frequency range.
	DSOUND	Turns off all sounds.
GO TO	GO TO *n*	Alternate form of GOTO.
***L**	*L	Turn line-indent on (Default).
	*L -	Turns line-indent off.
***F**	*F (or *F +)	Special mode for FOR..NEXT loops which corrects a bug in Atari BASIC. Seems that in Atari BASIC, an 'illegal' reverse loop like 'FOR X=2 TO 1:PRINT X:NEXT X' will execute once even though the condition is met initially (X is already greater than 1). Turbo BASIC fixes this bug, but leaves it available for Atari BASIC programs which may take advantage of it.
	*F -	Turns off the special FOR..NEXT mode to make Turbo BASIC act like Atari BASIC.
***B**	*B (or *B +)	Command which allows the break key to be trapped via the 'TRAP' command within a program.
	*B -	Turns off the special BREAK key mode.
--	--	Special form of REM which puts 30 dashes in a program listing.

LINE LABELS

#	# *name*	Assigns the current line number to the label name. This is a convenient way to get around the problem of renumbering when using variables as line numbers. Labels can be thought of as a special form of variable, as they occupy the variable name table along with the 'regular' variables. We also believe that the number of variables allowed has been increased from 128 to 256 to allow for the addition of these labels.
GO#	GO# *name*	Analagous to the GOTO command.

MODIFICATIONS

CLOSE	CLOSE	Close channels 1-7.
DIM	DIM *a(n)*	Will automatically assign a value of zero to all elements of the numeric array being dimensioned, and null characters to all elements of a string (The LEN is still variable, however, and initially zero).
GET	GET *name*	Wait for a key press, assign the value to name. Same as 'OPEN #7,4,0,"K:":GET #7,name:CLOSE #7'.
INPUT	INPUT *'text';a,b...*	Prints text as a prompt before asking for variable(s), same as Microsoft-BASIC.
LIST	LIST *n,*	List program from line n to end.
ON	ON *a* EXEC *n1,n2,...*	Variation of ON...GOSUB for procedures. N1, n2 and so on are names of procedures to be run.
	ON *a* GO# *n1,n2,...*	Similar to ON...GOTO except that line labels are used instead of line numbers.
POP	POP	This command now pops the runtime stack for all four types of loops.
PUT	PUT *n*	Same as 'PRINT CHR$(n)';
RESTORE	RESTORE *#name*	Restores the data line indicated by the label name.
RND	RND	Parentheses are no longer needed at the end of this command, but it will still work if they are there.
SOUND	SOUND	Turn off all sounds.
TRAP	TRAP *#name*	TRAPs to the line referenced by the label name.

TURBO BASIC FUNCTIONS

ARITHMETIC/LOGIC

HEX$	HEX$(*n*)	Convert n to hex string.
DEC	DEC(*a$*)	Convert hex string A$ to decimal.
DIV	*n* DIV *i*	Integer quotient of n/i.
MOD	*n* MOD *i*	Integer remainder of n/i.
FRAC	FRAC(*a*)	Fractional part of a.
TRUNC	TRUNC(*a*)	Truncates fractional part of a.
RAND	RAND(*n*)	Generates random number 0-n.
$	$*nnnn*	Allows input of hexidecimal numbers, but they are converted to decimal. Ex: 'FOR I=$0600 to $067F' = 'FOR I=1536 to 1663'.
&	*n* & *i*	8-bit boolean AND.
!	*n* ! *i*	8-bit boolean OR.
EXOR	*n* EXOR *i*	8-bit Exclusive-OR.

MEMORY

DPEEK	DPEEK(*m*)	Double-PEEK of m,m+1.
TIME	TIME	Time of day(numeric).
TIME$	TIME$	Time of day string, HHMMSS.
INKEY$	INKEY$	Returns last character typed.
INSTR	INSTR(*x$,a$*)	Returns relative location of start of string A$ within X$ (returns 0 if not found). The match must be exact; strings with the same letters but differences in case or type (normal or inverse) will not be found.
	INSTR(*x$,a$,i*)	i specifies the starting point of the search.
UINSTR	UINSTR(*x$,a$*)	Same as INSTR, does not distinguish between case or inverse characters. Ex: UINSTR("HeLlO","hello") returns 1.
	UINSTR(*x$,a$,i*)	Specifies optional starting point.
ERR	ERR	Value of last error number.
ERL	ERL	Line last error occurred at.

CONSTANTS

%0
%1
%2
%3

These four constants simply stand for the numbers 0-3, respectively. The difference with using these in a program is that 'X = 1' requires 10 bytes, whereas 'X = %1' only needs 4 (numbers require 7 bytes, 6 for the number plus an identifier preceeding it). It is always a good practice to make variables for numbers that are used more than three times in a program.

ADDITIONAL NOTES

1. Variable, Procedure and Label names may contain the underscore (_) character.
2. To print a double-quote (") in a text string, use two of them together, instead of the Atari BASIC method of using CHR$(34). Ex: "TEST";CHR$(34);"TEXT" becomes "TEST""TEXT" in Turbo-BASIC, both of which produce the output TEST"TEXT.
3. Upon initial boot-up, TURBO-BASIC looks for a BASIC file named AUTORUN.BAS. If it finds an AUTORUN.BAS file, it will automatically load and run this file.
4. A multiline IF is constructed like this:

```
10 IF X > 10
20 PRINT X-10
30 GO# TOO_BIG
40 ELSE
50 PRINT X
60 GO# X_IS_OK
70 ENDIF
```

Note also the use of line labels in the GOTO statements.

ERROR MESSAGES

Turbo-BASIC also prints out English descriptions of all errors, including several new ones for errors involving the new commands:

Error – 22 ?NEST = Loops not properly nested.
Error – 23 ?WHILE = WEND with no corresponding WHILE.
Error – 24 ?REPEAT = UNTIL with no corresponding REPEAT.
Error – 25 ?DO = LOOP with no corresponding DO.
Error – 26 ?EXIT = EXIT is outside a loop.
Error – 27 ?XPROC = Error executing PROC.
Error – 28 ?EXEC = ENDPROC with no corresponding EXEC.
Error – 29 ?PROC = Procedure does not exist.
Error – 30 ?# = Label does not exist.

Error 15 has been expanded to include an UNTIL which relates to a REPEAT which has been deleted.

THE TURBO COMPILER

Documentation and Operating Instructions
By Dave Arlington
Original Program by Frank Ostrowski

The TURBO COMPILER will increase the speed of your TURBO BASIC programs 3-5 times and regular Atari BASIC programs can be speeded up to 10-15 times faster. Unfortunately, like TURBO BASIC, the TURBO COMPILER will only run on the XL/XE series of computers and the finished compiled programs will also only run on the XL/XE series.

The TURBO COMPILER is very easy to operate and can be used with more than one disk drive, even Drive 8, the 130XE ramdisk. You will find several files on t TURBO BASIC/COMPILER disk. The files to use with the TURBO COMPILER are COMPILER.COM and RUNTIME.COM. Before you begin, you should prepare two disks. The first one should be a blank formatted disk with DOS.SYS and DUP.SYS written on it. The second disk should contain your BASIC program (TURBO or ATARI BASIC).

To begin, insert the TURBO COMPILER disk into your drive and turn your computer on. When it has finished loading you will be in TURBO BASIC. Type DOS to get to the DOS 2.5 menu. Choose DOS Option 'L', Load Binary File and load the file COMPILER.COM. After a short while, you will see a screen full of German. If you do not read German, do not worry, that is what this article is for! At any time that you are on this screen, you may reboot your system by hitting Control-R or you may return to DOS by hitting Control-D. You will be asked if you really want to do this. If you do, don't forget to 'J' for Ja and not 'Y' for Yes!!

To compile your programs, remove the TURBO COMPILER disk from the drive and insert the disk with the program you want to compile. (If you have two drives, insert your disk with the program to compile in Drive 2 and the blank DOS disk in Drive 1.) Press the number 1 key (Number 2 if you have two drives). A complete listing of all the files on that drive will appear on the screen with one highlighted in inverse video. Using the arrow keys, highlight the file you want to compile and hit RETURN. That's all there is to it!

At the top of the screen, you will see the line numbers fly by as the program is compiled. When it is finished, you will be prompted for a file name to save your compiled program under. It has to have an extender of CTB (for Compiled Turbo Basic). The program will not let you use any other extender. At this time, if you have one drive, you should remove the disk with your original program on and replace it with your blank DOS disk. If you want your compiled program to be an AUTORUN file, you should name it AUTORUN.CTB. Then the file will automatically load and run at bootup time.

There is one more step that you must do to get a completely runnable compiled program. Go back to DOS and copy the file RUNTIME.COM from the TURBO COMPILER disk to the disk with your compiled program on it. Rename the RUNTIME.COM file to the name AUTORUN.SYS. Your disk is now ready to go. Boot this disk and you are away! The TURBO COMPILER does not compile to runnable code, you must run the RUNTIME.COM file to run your compiled program. If you do not rename your compiled BASIC program to AUTORUN it may be loaded from the RUNTIME program by using 'L' at the on-screen prompt. At this prompt you may also re-run a program that has finished or quit to DOS]

One technical note on the TURBO COMPILER. If you are not familiar with other compilers for the ATARI computers, you should know that many have trouble compiling certain types of statements or insist that you organize your program structure in a certain way. So far very few of these problems have occured on any programs we have tested with the TURBO COMPILER. The only statements that wouldn't compile so far are the END and NEW statements. The only program structure that we have had trouble with is a FOR-NEXT loop that has two or more NEXTs for one FOR. For example:

```
10 FOR X=1 TO 5
20 IF X=3 THEN NEXT X
30 NEXT X
```

Other than those two examples, we have had no problems compiling any type of program, either TURBO or ATARI BASIC. If the compiler does encounter a problem it will indicate the type of fault and the line number. Simply check it out and try to rewrite that part of the program in a different way. Don't be discouraged, no compiler can cope with *every* program!

TEXT TIDIER 1.0

by Dave Yearke

Text Tidier was designed to eliminate much of the editing that goes into processing files downloaded from such services as CompuServe. These files often have excess spaces for justification, carriage returns every line, and all kinds of other annoying things that must be weeded out. The most bothersome types of text files are those which use the Control-J Control-M combination for linefeed and carriage return. For MS-DOS machines that's fine, but our Ataris use a totally different character.

In its simplest form, Tidier strips out linefeeds (CTRL-J) and turns ASCII carriage returns (CTRL-M) into ATASCII returns (code 155). It also breaks down the file into smaller pieces that will fit into a word processor like AtariWriter or PaperClip. I've downloaded files that were over 100K long, far too big to fit into any word processor for the Atari!

When you run the program, the first thing it asks for is the maximum file size for the output files. 12,000 is pretty good for AtariWriter, although this depends on what DOS is being used, whether a printer driver is loaded, and so on. For PaperClip, I've found that on my XL about 24,000 bytes can be loaded, although I would recommend about 20,000 so you have room to edit the file. Next it asks for the Input file. Put the disk in the drive and enter the name. If you don't enter a device like "D:" it will default to drive one. All Output files will be named 'SPLIT.0xx', where 'xx' is the number of the file being written, starting at zero. Be careful not to overwrite any existing files with the same name! Now come the formatting parameters. When it asks 'CONVERT RETURN TO SPACE (Y/N)' type 'Y' if you want all carriage returns to be converted. This is nice if you want to reformat the text in your word processor. This option only converts single returns; if it encounters more after the first one it will leave them alone because it assumes that it is a break between two paragraphs. The next option, 'TIDY UP SPACING (Y/N)', is probably the most powerful. It will get rid of multiple spaces between words, excess spaces before return characters, convert ASCII Tab characters (CTRL-I) to 5 spaces, and will make sure that two spaces follow all periods, colons, question marks, and exclamation points, and that one space follows all commas, even if it has to add them. One note: it will not add spaces to periods which have no spaces after them, because it can't tell if it is part of a decimal number.

The next option, 'PROMPT BEFORE WRITING (Y/N)', should get a 'Y' response if you need to switch disks between reading and writing, and an 'N' response if you want to let it go without interruption. The 'BREAK FILES ON SPACE (Y/N)' simply assures that a word won't be split over two files (this may not work if the last word before writing is over 15 characters long, but that's pretty rare). The last option is a convenience to those with the AtariWriter or PaperClip word processors. If you select either 'A' or 'P' at the prompt, it will write the 'chain next file' command at the end of all files except the last one. In addition, it will put a paragraph symbol where it thinks it's appropriate, and convert the ASCII formfeed character (CTRL-L) to its AW or PC equivalent. If you select 'N' for 'neither', then the paragraphs will be indented five spaces and formfeeds will be left alone.

By the way, the program only allows character codes within the range 26-127, plus CR (13) and FF(12). Characters higher than 127 have the high bit stripped to make them 'normal;' while control characters are ignored. If you want to allow inverse and control characters (although this can be a potential problem with true ASCII files; control characters can have bizarre meanings, and true ASCII does not allow character codes higher than 127. If you get any, it's probably a transmission error), change line 215 to:

```
215 REPEAT :GET #1,N:UNTIL (N< >LF)
```

This will only filter linefeed characters.

This program was written out of necessity, but I also had a lot of fun doing it. I hope that you also find the program useful.

OTHER PROGRAMS ON THIS DISK:

MAGIC.TUR – A Koala type drawing program written in Turbo Basic.

PMMOVE.TUR and PUTGET.TUR – Two demo programs that show off some of TBs power.

COMMAND SUMMARY

--	--
!	n ! i
#	# name
$	$nnnn
%0	Constant
%1	Constant
%2	Constant
%3	Constant
%PUT	%PUT #n,a
%GET	%GET #n,A
&	n & i
***L**	*L
	*L -
***F**	*F (or *F +)
	*F -
***B**	*B (or *B +)
	*B -
BGET	BGET #n,adr,len
BLOAD	BLOAD "D:name"
BPUT	BPUT #n,adr,len
BRUN	BRUN "D:name"
CIRCLE	CIRCLE x,y,r
	CIRCLE x,y,r,r2
CLOSE	CLOSE
CLS	CLS
	CLS #6
DEC	DEC(a$)
DEL	DEL n,i
DELETE	DELETE "D:name"
DIM	DIM a(n)
DIR	DIR
	DIR "Dn:*.*"
DIV	n DIV i
DO	DO
DPEEK	DPEEK(m)
DPOKE	DPOKE m,v
DSOUND	DSOUND n,f,d,v
	DSOUND
DUMP	DUMP
	DUMP name
ELSE	ELSE
ENDIF	ENDIF
ENDPROC	ENDPROC
ERL	ERL
ERR	ERR
EXEC	EXEC name
EXIT	EXIT
EXOR	n EXOR i
FCOLOR	FCOLOR n
FILLTO	FILLTO x,y
FRAC	FRAC(a)
GET	GET name
GO#	GO# name
GO TO	GO TO n
HEX$	HEX$(n)
INKEY$	INKEY$
INPUT	INPUT 'text';a,b...
INSTR	INSTR(x$,a$)
	INSTR(x$,a$,i)
LIST	LIST n,
LOCK	LOCK "D:name"
LOOP	LOOP
MOD	n MOD i
MOVE	MOVE m,m1,m2
-MOVE	-MOVE m,m1,m2
ON	ON a EXEC n1,n2,...
	ON a GO# n1,n2,...
PAINT	PAINT x,y
PAUSE	PAUSE n
POP	POP
PROC	PROC name
PUT	PUT n
RAND	RAND(n)
RENAME	RENAME "D:old,new"
RENUM	RENUM n,i,j
REPEAT	REPEAT
RESTORE	RESTORE #name
RND	RND
SOUND	SOUND
TEXT	TEXT x,y,a$
TIME	TIME
TIME$	TIME$
TRACE	TRACE
	TRACE -
TRAP	TRAP #name
TRUNC	TRUNC(a)
UINSTR	UINSTR(x$,a$)
	UINSTR(x$,a$,i)
UNLOCK	UNLOCK "D:name"
UNTIL	UNTIL <c>
WEND	WEND
WHILE	WHILE <c>