```
1790    FOR SLEEP% = 1 TO 100: NEXT SLEEP%
1800    OUT LCR,SD%
1810 RETURN
1820 ' Close connections
1830    CLOSE
1840 '  PAUSE% = 0
1850 RETURN
1860 ' Draw menu screen
1870    CLS
1880    LOCATE 1,1,1: ROW% = 5
1890    PRINT CAPTION$: PRINT
1900    FOR I% = 1 TO NSETTINGS%
1910       LOCATE ROW%, 1: PRINT I%; NM$[I%]
1920       LOCATE ROW%, 23: PRINT "["+VL$[I%
1930       ROW% = ROW% + 1: IF ROW% = 8 THEN
1940    NEXT I%
1950    GOSUB 1970
1960 RETURN
1970 ' Draw key line
1980    IF STATE% THEN KEY1$="Menu" ELSE KE
1990    LOCATE 25,1: PRINT "f1=" + KEY1$ +
2000 RETURN
Ok
```

PC-BASIC documentation

Version 2.0.1

1. PC-BASIC 2.0.1

A free, cross-platform emulator for the GW-BASIC family of interpreters.

PC-BASIC is a free, cross-platform interpreter for GW-BASIC, Advanced BASIC (BASICA), PCjr Cartridge Basic and Tandy 1000 GWBASIC. It interprets these BASIC dialects with a high degree of accuracy, aiming for bug-for-bug compatibility. PC-BASIC emulates the most common video and audio hardware on which these BASICs used to run. PC-BASIC runs plain-text, tokenised and protected .BAS files. It implements floating-point arithmetic in the Microsoft Binary Format (MBF) and can therefore read and write binary data files created by GW-BASIC.

This is the documentation for **PC-BASIC 2.0.1**, last updated *2018-12-01 12:20:49*.
It consists of the following documents:

- **Quick Start Guide**, the essentials needed to get started
- **User's Guide**, in-depth guide to using the emulator
- **Configuration Guide**, settings and options
- **Language Guide**, overview of the BASIC language by topic
- **Language Reference**, comprehensive reference to BASIC
- **Technical Reference**, file formats and internals
- **Developer's Guide**, using PC-BASIC as a Python module

Table of Contents

Statements

2. Quick Start Guide

This quick start guide covers installation and elementary use of PC-BASIC. For more information, please refer to the PC-BASIC documentation.

If you find bugs, please open an issue on GitHub. It would be most helpful if you could include a short bit of BASIC code that triggers the bug.

2.1. Installation

PC-BASIC desktop installers for Windows, Mac, and Linux can be downloaded from <u>GitHub</u>.

Python users can obtain the PC-BASIC package from <u>PyPI</u> through `pip install pcbasic`.

2.2. BASIC survival kit

PC-BASIC has a 1980s-style interface operated by executing typed commands. There is no menu, nor are there any of the visual clues that we've come to expect of modern software.

A few essential commands to help you get around:

Command	Effect
`LOAD "PROGRAM"`	loads the program file named `PROGRAM.BAS` into memory
`LIST`	displays the BASIC code of the current program
`RUN`	starts the current program
`SAVE "PROGRAM",A`	saves the current program to a text file named `PROGRAM.BAS`
`NEW`	immediately deletes the current program from memory
`SYSTEM`	exits PC-BASIC immediately, discarding any unsaved program

Use one of the key combinations `Ctrl+Break`, `Ctrl+Scroll Lock`, `Ctrl+C` or `F12+B` to interrupt a running program.

2.3. Program location

If started through the start-menu shortcut, PC-BASIC looks for programs in the shortcut's start-in folder.

- On **Windows**, this is your `Documents` folder by default.
- On **Mac** and **Linux** this is your home directory `~/` by default.

If started from the command prompt, PC-BASIC looks for programs in the current working directory.

See the documentation on accessing your drives for more information.

2.4. External resources

The following pages have GW-BASIC program downloads, lots of information and further links.

- KindlyRat's archived Geocities page has a number of classic games and utilities.
- PeatSoft provides GW-BASIC documentation, utilities and some more games.
- Leon Peyre has a nice collection of GW-BASIC programs, including the (in)famous first IBM PC game `DONKEY.BAS`.
- Brooks deForest provides his amazing Tandy GW-BASIC games, all released into the public domain.
- TVDog's Archive is a great source of information and GW-BASIC programs for the Tandy 1000.
- Phillip Bigelow provides scientific programs written in GW-BASIC.
- Gary Peek provides miscellaneous GW-BASIC sources which he released into the public domain.
- S.A. Moore's Classic BASIC Games page provides the BASIC games from David Ahl's classic book.
- Joseph Sixpack's Last Book of GW-BASIC has lots of office and utility programs, including the PC-CALC spreadsheet.
- Thomas C. McIntyre's GeeWhiz Collection has business applications, games and reference material.
- cd.textfiles.com has tons of old shareware, among which some good GW-BASIC games.
- PC-SIG Library 8th Edition CD-ROM (April 1990) hosts many disk images and features a "live" DOS box.
- The MS-DOS source code, as released by Microsoft, contains the original IBM PC Samples BASIC demos and games.
- Antonis.de focusses on Q-BASIC but has a number of GW-BASIC resources, tutorials, e-books etc.
- Microsoft GW-BASIC Manual: an online version of the original User' Guide and User's Reference.
- BASIC Training: a GW-BASIC tutorial by Steve Estvanik.
- Blast Off With BASIC: a GW-BASIC tutorial by Brian R. Page.
- The BLUE Book about GW-BASIC and QuickBASIC: the excellent "BASIC Language User Essay" by Thomas C. McIntire is full of expert knowledge about GW-BASIC.

3. User's guide

3.1. The working environment

The first thing you'll see when starting PC-BASIC is the working environment. Like GW-BASIC, but unlike practically all modern compilers and interpreters, PC-BASIC's working environment serves both as a development environment and as a canvas on which to execute BASIC commands directly. With a few exceptions, practically all commands that can be run in the working environment can be used in a program, and vice versa.

The default PC-BASIC screen has 25 rows and 80 columns. The 25th row is used by PC-BASIC to show keyboard shortcuts, which means you can't use it to type on. In some video modes, there are only 40 or 20 columns.

Logical lines exceed the width of the physical row: if you keep typing beyond the screen width, the text will wrap to the next line but PC-BASIC will still consider it part of the same line. A logical line can be at most 255 characters long; if you type more than 255 characters, it will ignore the remainder. A line can also be wrapped by a line-feed, entered with `Ctrl` + `Enter` .

If you press `Enter` , PC-BASIC will attempt to execute the logical line on which the cursor is placed as a command. When the command is executed correctly, PC-BASIC will display the prompt `Ok` . If there is an error, it will display an error message followed by `Ok` . If the line starts with a number, it will be stored as a program line. No prompt is displayed.

Special keys

The following keys have a special effect in the working environment:

Key	Description
`↑` or `Ctrl`+`6`	Move the cursor up, except at the top row.
`↓` or `Ctrl`+`-`	Move the cursor down, except at row 24.
`←` or `Ctrl`+`]`	Move the cursor left. The left edge of the screen wraps around, except at the top row.
`→` or `Ctrl`+`/`	Move the cursor right. The right edge of the screen wraps around, except at row 24.
`Ctrl`+`←` or `Ctrl`+`B`	Move to the first letter of the previous word. Words consist of letters `A-Z` and figures `0-9`.
`Ctrl`+`→` or `Ctrl`+`F`	Move to the first letter of the next word.
`Tab` or `Ctrl`+`I`	Move the cursor to the next tab stop. Tab stops are 8 columns wide.
`Backspace` or `Ctrl`+`H`	Delete the character left of the cursor, shift all further characters on the logical line one position to the left and change the attributes of those characters to the current attribute. At the left edge of the screen, this does the same as `Del`.
`Del` or `Ctrl`+`Backspace`	Delete the character at the cursor and shift all further characters one position to the left, changing attributes to current.
`Esc` or `Ctrl`+`[`	Delete the current logical line.
`Ctrl`+`End` or `Ctrl`+`E`	Delete all characters from the cursor to the end of the logical line.
`Ctrl`+`Break` or `Ctrl`+`C` or `Ctrl`+`Scroll Lock`	Jump to the first column of the next line, without executing or storing the line under the cursor.
`Enter` or `Ctrl`+`M`	Execute or store the current logical line. The complete line on the screen is considered part of the command, even if you did not type it. A line starting with a number is stored as a program line.
`End` or `Ctrl`+`N`	Move the cursor to the first position after the end of the logical line.
`Home` or `Ctrl`+`K`	Move the cursor to the top left of the screen.
`Ctrl`+`Home` or `Ctrl`+`L`	Clear the screen and move the cursor to the top left of the screen.
`Ctrl`+`Enter` or `Ctrl`+`J`	Move to the first column of the next line, connecting the two lines into one logical line.
`Ctrl`+`G`	Beep the speaker.
`Pause` or `Ctrl`+`Num Lock`	Pause. Press another key to resume. The latter key press will not be detected by BASIC.
`Ctrl`+`Prt Sc`	Toggle echoing screen output to the printer (or other device attached to `LPT1:`).

`Shift` + `Prt Sc`	Print the screen.
`Ins` or `Ctrl` + `R`	Toggle *insert mode*. In insert mode, characters are inserted rather than overwritten at the current location. If insertion causes the line to extend the physical screen width, the logical line extends onto the next line. Arrow keys exit insert mode.

When a program is started, the commands in the program are followed until the program quits and returns to direct mode or until user input is required. When a program is running, a few keys have immediate effect:

`Pause` or `Ctrl` + `Num Lock`	Pause execution. Press another key to resume.
`Ctrl` + `Break` or `Ctrl` + `Scroll Lock`	Stop execution and return to direct mode. A `Break` message is printed.
`Ctrl` + `C`	If `ctrl-c-break`=`True`: stop execution and return to direct mode. A `Break` message is printed.

If user input is required by the statements `INPUT`, `LINE INPUT`, or `RANDOMIZE`, most keys have the same effect as in direct mode. The following keys have a different effect:

`Ctrl` + `Break` or `Ctrl` + `C` or `Ctrl` + `Scroll Lock`	Stop execution and return to direct mode. A `Break` message is printed.
`Enter`	Finish input and return to the previous mode.

Keyboard shortcuts

The function keys and the alt key can be used as keyboard shortcuts for some keywords. The default values for the function keys are:

`F1`	`LIST`
`F2`	`RUN` `Enter`
`F3`	`LOAD"`
`F4`	`SAVE"`
`F5`	`CONT` `Enter`
`F6`	`,"LPT1:"` `Enter`
`F7`	`TRON` `Enter`
`F8`	`TROFF` `Enter`
`F9`	`KEY` `Space`
`F10`	`SCREEN 0,0,0` `Enter`

The function key shortcuts can be redefined with the `KEY` statement. The shortcuts are displayed at the bottom of the screen.

The following keywords can be entered with `Alt` +first letter. The `Alt` shortcuts cannot be redefined.

AUTO	BSAVE	COLOR	DELETE	ELSE	FOR
GOTO	HEX$	INPUT	KEY	LOCATE	MOTOR
NEXT	OPEN	PRINT	RUN	SCREEN	THEN
USING	VAL	WIDTH	XOR		

Alternative keys

In PC-BASIC, the `F12` key can be used to enter special keys that are not present on some keyboards.

`F12` + `B`	`Ctrl` + `Break`
`F12` + `P`	`Pause`
`F12` + `C`	`Caps Lock`
`F12` + `N`	`Num Lock`
`F12` + `S`	`Scroll Lock`
`F12` + `H`	`Print Screen`

The `F12` key can also be used in combination with the regular number keys and arrow keys to enter numbers from the numeric keypad. The `F12` combinations are not present in GW-BASIC.

Furthermore, as in GW-BASIC, the `Alt` key can be used to enter characters by their code points (ASCII values). This is done by pressing the `Alt` key and typing the code point as a decimal value on the numeric keypad, then releasing the `Alt` key.

Clipboard operations

Unlike in GW-BASIC, you can copy and paste text to the clipboard. This can be done with the mouse or with the `F11` key.

Operating the clipboard with the mouse works in the style of X11: Left button is select and copy; middle button is paste.

The following keyboard combinations also operate the clipboard:

`F11` + `↑` `↓` `←` `→`	Select a screen region.
`F11` + `A`	Select all.
`F11` + `C`	Copy to clipboard.
`F11` + `V`	Paste from clipboard.

Emulator control keys

In PC-BASIC, `F11` + `F` toggles fullscreen mode.

Compatibility

Some key combinations may have a different effect than described above, depending on the operating system and the choice of interface to use with PC-BASIC.

- Certain key combinations will be interpreted by the operating system or window manager and cause special actions. For example, on most systems, `Alt`+`F4` will terminate PC-BASIC unless the `prevent_close` option is set; `F1` may open your operating system's help system. It may be possible to avoid some of these effects by using the graphical interface in full-screen mode.
- In the command-line interface on Windows, `Ctrl`+`C` terminates PC-BASIC.
- In the command-line interface on Linux and Mac, `Ctrl`+`D` terminates PC-BASIC.

3.2. Programs and files

PC-BASIC can hold one BASIC program at a time in memory. To enter a program line, start with a *line number* and enter BASIC commands after that. The maximum length of a program line is 255 characters, including the line number and any spaces. The program line will not be immediately executed, but stored in the program. Program lines are sorted by line number, so that line `10` is executed before line `20` . All program lines must have a line number. Line numbers range from `0` to `65535` inclusive. It is not possible to enter a line number higher than `65529` , but these can exist in loaded programs. Within one program line, statements are separated by colons `:` .

To run the program, type the command `RUN` . PC-BASIC will now execute all program lines in order inside the working environment. You cannot move the cursor around or enter commands while the program is running. If and when the program finishes, it will return control of the working environment to you. You can interrupt a program at any time by using one of the key combinations `Ctrl`+`Break` or `Ctrl`+`Scroll Lock` . The program will stop immediately, print a `Break` message and return control to you.

In GW-BASIC, you can *not* use `Ctrl`+`C` to interrupt a running program. However, many modern keyboards do not have a `Break` or `Scroll Lock` key, which would make it impossible to interrupt a program that does not exit. Therefore, by default, PC-BASIC treats `Ctrl`+`C` as if it were `Ctrl`+`Break` . Set the option `ctrl-c-break=False` if you prefer the GW-BASIC style behaviour. When using the text-based or command-line interface, this option is ignored.

A program can be stored on a drive by using the `SAVE` command, in one of three ways:

1. Plain text, readable by any text editor: `SAVE "MYPROG",A`
2. Tokenised, taking up less storage space: `SAVE "MYPROG"`
3. Protected, which is an encrypted format: `SAVE "MYPROG",P`

In all three cases, the program will be written to the current working directory with the name `MYPROG.BAS` .

PC-BASIC can read and write Protected files created by GW-BASIC. Unlike GW-BASIC, however, it does not disable accessing the unencrypted contents of the file. The encryption used by GW-BASIC has been broken many decades ago, so Protected mode offered little protection anyway; disallowing access is a small security hazard as it would allow someone to send you a program that you cannot inspect before running it. However, it is possible to disable access of protected files by enabling the option `hide-protected` .

You can read a program file into memory with `LOAD` `"MYPROG"` . This will erase the program currently in memory and replace it with the one read from the current working directory. To access files in a different directory, specify a path from the current directory. The path specification follows DOS conventions. The only valid path separator is the backslash `\` . For example, `LOAD` `"PROGRAMS\MYPROG"` .

You can load or run a program immediately on starting PC-BASIC by using the `load` or `run` options. For example,

```
pcbasic --run=MYPROG.BAS
```

The arguments to these options can be provided as PC-BASIC paths or as paths in the standard form for your operating system.

PC-BASIC can be used to convert between the three program formats: either by loading the program and saving in your desired format, or from the command line using the `convert` option. To convert a tokenised or protected file to plain text you could use, for example:

```
pcbasic --convert=A PROGRAMP.BAS PROGRAMA.BAS
```

Accessing your drives

PC-BASIC emulates DOS disk devices, which are referred to by drive letters such as `z:` . One of the drive letters is the *current device*.

On **Windows**:

- By default, PC-BASIC disk devices will agree with Windows drive letters at the start of the PC-BASIC session.
- If PC-BASIC is started from the start menu shortcut, the current device will be your *Documents* folder (or *My Documents* on some versions of Windows). You can change this location by setting the shortcut's *Start In* folder.
- If PC-BASIC is started from the command prompt, the current device will be set to the current working directory of the command prompt.
- If PC-BASIC's current device or *Start In* folder is changed to a system folder such as `C:\Program Files\PC-BASIC` , Windows will move files written there to `%LocalAppData%\VirtualStore` instead. This is best avoided.
- Note that *PC-BASIC's DOS disk devices are not the same thing as Windows drive letters*. The device `c:` on PC-BASIC is *not* always your Windows `c:` drive. By default, Windows drive letters are mapped to PC-BASIC devices at the start of the PC-BASIC session. However, if you use the `mount` option; or if Windows drive letters change while PC-BASIC is running (through e.g. `net use` or *Map Network Drive* operations), they will no longer agree.

On **other systems**:

- By default, `z:` will point to the current working directory from where PC-BASIC was started. It will be the current device.
- If started from a menu or app package, this will usually be your home directory `~` .

This current device is where files will be saved to and loaded from in BASIC if you do not specify another device. You can change the current device using the `current-device` option in the <u>configuration file or on the command prompt</u>.

You can map drives and other file system locations as PC-BASIC devices by using the `mount` option. For example, on Windows, the option

```
mount=A:C:\Users\Me\BasicFloppy
```

will make the folder `C:\Users\Me\BasicFloppy` available as PC-BASIC's `A:` device. On other platforms, an example mount option could look like

```
mount=A:/home/me/BasicFloppy
```

which would make the directory `/home/me/BasicFloppy` available as PC-BASIC's `A:` device.

PC-BASIC uses DOS conventions for filenames and paths. These are subtly different from Windows short filename conventions and not-so-subtly different from Unix conventions. This may lead to surprising effects in the presence of several files that match the same DOS name. To avoid such surprises, it's best to run PC-BASIC in a working directory of its own and use all-caps 8.3 format for all files.

Compatible BASIC files

Many BASIC dialects use the same extension `.BAS`, but their files are not compatible. PC-BASIC runs GW-BASIC program files only. Some tips to recognise GW-BASIC programs:

- GW-BASIC files stored as text are plain text files with line numbers.
- Tokenised files are binary files that start with magic byte `&hFF`.
- Protected files are binary files that start with magic byte `&hFE`.

In particular, QBASIC files (which have no line numbers) and QuickBASIC files (magic byte `&hFC`) will not run.

PC-BASIC will accept both DOS and Unix newline conventions for programs stored as plain text. This behaviour is different from GW-BASIC, which only accepts text files with `CR LF` line endings. As a consequence, in exceptional cases where a program line is continued through `LF` correct GW-BASIC text files may not be loaded correctly. If you encounter such a case, use the `soft-linefeed` option to enable GW-BASIC behaviour. If `soft-linefeed` is enabled, text files in standard Unix format (`LF` line endings, no end-of-file character) will fail to load, as they do in GW-BASIC. On Linux or Mac, use a utility such as `unix2dos` to convert programs saved as text files before loading them. When saving as text, PC-BASIC always uses `CR LF` line endings and `&h1A` at end-of-file.

Packages

PC-BASIC can run packaged programs. A package is simply a directory or zip archive. The directory or zipfile contents will be loaded as the current working directory. If a configuration file named `PCBASIC.INI` is present inside this directory, its settings are loaded; usually, one of those settings will be a `run` argument linking to a BASIC program enclosed in the archive or directory. PC-BASIC will recognise zipfiles regardless of their extension. A suggested extension for PC-BASIC packages is `.BAZ`. Packages are a convenient choice if a program needs to change many PC-BASIC options to function as desired, or if it needs a particular working directory setup.

Zipfile packages are unpacked to a temporary directory each time they are loaded. The temporary directory is removed when PC-BASIC closes. With zipfile packages, it is therefore not possible to save files and re-open them on the next run of the package.

Cassette tapes

The `CAS1` device interfaces with the cassette tape emulator. Tapes were never very popular on the IBM PC, and indeed only available with the original PC and the PCjr. There are not many IBM PC cassettes in the wild. However, should you come across one, all you have to do to read it with PC-BASIC is record it into a `.WAV` (RIFF WAVE) file and attach it to the `CAS1:` device with the `cas1=WAV:`*filename* option. You can also generate your own tape images and store your programs on it. `WAV` files generated by PC-BASIC are large but very easily compressed in a `ZIP` archive; this works better and leads to smaller files than transcoding to a lossy audio format like `MP3`.

As an alternative to `.WAV`, you can store tapes in `CAS` format. This is simply a bit-dump of the tape and is interchangeable with tape images for the PCE IBM PC emulator.

Previous versions of PC-BASIC included support for BASICODE cassettes; this has been discontinued in favour of a separate BASICODE decoder. Use this decoder to convert the BASICODE program to PC-BASIC format before loading it into PC-BASIC.

Security

PC-BASIC makes some default choices with basic security in mind, but does not sandbox its programs in any meaningful way. BASIC programs have more or less full access to your computer. You should treat them with the same caution as you would shell scripts or binaries. Therefore, do not run a program from the internet that you have not inspected first using `LIST` or

```
pcbasic --convert=A filename
```

on the command line. You wouldn't just download an executable from the internet and run it either, right?

3.3. Connecting to peripherals

Printing

You can print from PC-BASIC programs by accessing the `LPT1:` device. PC-BASIC will send the output to your operating system's default printer, unless you change the `lpt1=` option. To print through a printer named `MyPrinter`, set **`lpt1=PRINTER:MyPrinter`**. You can also attach printers to the `LPT2:` and `LPT3:` devices.

The output will be sent to the printer when one of the following happens: a file open to `LPT1:` is closed, a program terminates, or PC-BASIC is closed. If you prefer, you can instead send every page separately to the printer by setting `lpt1=PRINTER:MyPrinter:page`. You can even send every line separately, but this only makes sense on a tractor-fed printer (as was common in GW-BASIC's heyday).

It's easy to print to a file instead of a printer: set **`lpt1=FILE:output.txt`** to send all `LPT1:` printer output to the text file `output.txt`.

The printing statements `LPRINT` and `LLIST` always send their output to PC-BASIC's `LPT1:` device.

The presentation of printed documents is left to your operating system: it will be the default presentation of text files. If you wist to change the way documents are printed, please refer to your OS's settings. On Windows, for example, text files are printed by `notepad.exe` and changing the default settings in that application will change the way PC-BASIC documents are printed. You will need to set a printer font that includes the characters you need to print. On Unix systems, PC-BASIC will use the `paps` utility if it is available; this will automatically select fonts that support the characters you need.

Serial and parallel ports

PC-BASIC provides the serial devices `COM1:` and `COM2:`. To make use of these, you need to attach them to a communications port on your computer with the `com1=` or `com2=` option. To attach to the first physical serial port, set **`com1=PORT:0`** (or, alternatively, **`com1=PORT:COM1`** on Windows or **`com1=PORT:/dev/ttyS0`** on Linux). If you do not have a serial port, you can emulate one by sending the communications over a network socket: set **`com1=SOCKET:localhost:7000`** and all `COM1:` traffic will be sent through socket `7000`.

To access a parallel port, attach it to one of `LPT1:`, `LPT2:` or `LPT3:`. For example, set **`lpt2=PARPORT:0`** to attach your computer's first parallel port to `LPT2:`.

3.4. Changing the interface

Emulation targets

By default, PC-BASIC emulates GW-BASIC on a system with VGA video capabilities. However, it can emulate several other setups, which differ from each other in terms of video and audio capacity, fonts, memory size, as well as available BASIC syntax. The easiest way to set the emulation target is by using a `preset` option. For example, run `pcbasic --preset=pcjr`. Other available emulation target presets are:

Preset	Emulation target
pcjr	IBM PCjr with Cartridge BASIC, including PCjr video and 3-voice sound capabilities and extended BASIC syntax.
tandy	Tandy 1000 with GW-BASIC, including Tandy video and 3-voice sound capabilities and extended BASIC syntax.
olivetti	Olivetti M24 or AT&T PC 6300.
cga	IBM or compatible with Color/Graphics Adapter and a composite monitor. This enables composite colorburst emulation.
ega	IBM or compatible with Extended Graphics Adapter.
vga	IBM or compatible with Video Graphics Array.
mda	IBM or compatible with Monochrome Display Adapter and green-tinted monochrome monitor.
hercules	IBM compatible with Hercules Graphics Adapter and green-tinted monochrome monitor.
strict	Choose strict compatibility with GW-BASIC over convenience, security, rhyme or reason.

Presets are groups of options that are defined in the default configuration file. You can create your own presets by creating a header in your private configuration file with the name of the new preset, followed by the options you want to apply. For example, if you define:

```
[my_preset]
video=vga
syntax=pcjr
```

you can now run `pcbasic --preset=my_preset` to start an emulation of a hypothetical machine with a VGA video card running PCjr Cartridge BASIC.

GW-BASIC compatibility features

PC-BASIC aims for a very high level of compatibility with GW-BASIC. However, some compatibility features are disabled by default for convenience or security reasons. These features can be switched on using individual command-line options. The highest level of compatibility with GW-BASIC can be attained by setting `preset=strict`, which switches off all convenience and security features that cause differences with GW-BASIC.

Codepages

PC-BASIC supports a large number of legacy codepages that were common at the time GW-BASIC was popular, including double-byte character set codepages used for Chinese, Japanese and Korean. You can select your codepage by using the `codepage=` option. For example, `codepage=936` selects the GBK codepage commonly used on the Chinese mainland. PC-BASIC will load and save all program files as if encoded in the codepage you select.

It is also possible to load and save programs in the nowadays common UTF-8 standard format, by enabling the `utf8` option. When `utf8` is enabled, plain-text program source will be saved and loaded in standard UTF-8 encoding. Please note that you will still need to select a codepage that provides all the Unicode characters that your program needs.

Note that PC-BASIC does not implement the following features relevant to some of these codepages:

Bidirectional text

All text is printed left-to-right independent of the codepage selected. To write strings in a language that is written right-to-left, the logical character sequence must be inverted so that the order appears correct visually. While this is inconvenient, it is in line with the behaviour of GW-BASIC. This affects code pages marked with *B* in the table.

Combining characters

PC-BASIC recognises single-byte code points (where each glyph shows on a single cell on the screen) and double-byte code points (where a single glyph takes up two cells on the screen). Combining characters (such as the combining diacritics of codepages `874` and `1258`) are therefore not shown correctly: instead of being combined with their preceding base character as a single combined glyph, such combinations will be shown as separate glyphs. Where available, alternative codepages with precomposed characters will give better results. This affects code pages marked with *C* in the table.

The following codepages are available. PC-BASIC uses the Microsoft OEM codepage number where this is unambiguous. The code pages are expected to agree with Microsoft sources for the ranges `&h80` – `&hFF`. Ranges `&h00` – `&h1F` and `&h7F` are implemented as the IBM Special Graphic Characters where some code page sources will list these as the corresponding control characters. For unofficial codepages and those with conflicting numbering, codepage names are used instead of numbers.

codepage_id	Codepage	Languages	Notes
437	DOS Latin USA	English	
720	Transparent ASMO	Arabic	*B*
737	DOS Greek	Greek	
775	DOS Baltic Rim	Estonian, Latvian and Lithuanian	
806	IBM-PC Devanagari	Hindi	
850	DOS Latin 1	Western European languages	
851	DOS Greek 1	Greek	
852	DOS Latin 2	Central European languages	
853	DOS Latin 3	Southern European languages	
855	DOS Cyrillic 1	Serbian, Macedonian and Bulgarian	
856	DOS Hebrew	Hebrew	*B*
857	DOS Latin 5	Turkish	
858	DOS Latin 1 with Euro	Western European languages	
860	DOS Portuguese	Portuguese	
861	DOS Icelandic	Icelandic	
862	DOS Hebrew	Hebrew	*B*
863	DOS Canadian French	French	
864	DOS Arabic	Arabic	*B*
865	DOS Nordic	Danish and Norwegian	
866	DOS Cyrillic 2	Russian	
868	DOS Urdu	Urdu	*B*
869	DOS Greek 2	Greek	
874	TIS-620	Thai	*C*
932	Shift-JIS (variant)	Japanese	
934	DOS/V Korea	Korean	
936	GBK; GB2312/EUC-CN superset	Simplified Chinese	
938	DOS/V Taiwan	Traditional Chinese	
949	IBM-PC Korea KS; EUC-KR superset	Korean	
950	Big-5 (variant)	Traditional Chinese	
1258	Vietnamese	Vietnamese	*C*

`alternativnyj`	GOST Alternativnyj Variant	Russian	
`armscii8a`	ArmSCII-8a; FreeDOS cp899	Armenian	
`big5-2003`	Big-5 (Taiwan 2003)	Traditional Chinese	
`big5-hkscs`	Big-5 (Hong Kong 2008)	Traditional Chinese	
`georgian-academy`	Academy Standard	Georgian	
`georgian-ps`	Parliament Standard	Georgian	
`iransystem`	Iran System	Persian	*B*
`iscii-as`	ISCII Assamese	Assamese	*C*
`iscii-be`	ISCII Bengali	Bengali	*C*
`iscii-de`	ISCII Devanagari	Hindi, Marathi, Sanskrit, Konkani, Nepali	*C*
`iscii-gu`	ISCII Gujarati	Gujarati	*C*
`iscii-ka`	ISCII Kannada	Kannada	*C*
`iscii-ma`	ISCII Malayalam	Malayalam	*C*
`iscii-or`	ISCII Oriya	Oriya	*C*
`iscii-pa`	ISCII Punjabi	Punjabi	*C*
`iscii-ta`	ISCII Tamil	Tamil	*C*
`iscii-te`	ISCII Telugu	Telugu	*C*
`kamenicky`	Kamenický; cp895	Czech	
`koi8-r`	KOI8-R	Russian	
`koi8-ru`	KOI8-RU	Ukrainian, Belarusian, Russian	
`koi8-u`	KOI8-U	Ukrainian, Russian	
`mazovia`	Mazovia; cp667, 991, 790	Polish	
`mik`	MIK, FreeDOS cp3021	Bulgarian	
`osnovnoj`	GOST Osnovnoj Variant	Russian	
`pascii`	PASCII	Kashmiri, Persian, Sindhi, Urdu	*B*
`ruscii`	RUSCII	Ukrainian, Russian	
`russup3`	Cornell Russian Support for DOS v3	Russian	
`russup4ac`	Exceller Software Russian Support for DOS v4 Academic	Russian	
`russup4na`	Exceller Software Russian Support for DOS v4 Non-Academic	Russian	
`viscii`	VISCII, FreeDOS cp30006	Vietnamese	

You can add custom codepages to PC-BASIC, by adding a file with its mapping to Unicode to the `codepage/` directory.

Fonts

PC-BASIC emulates the distinctive raster fonts of IBM-compatible machines. The ROM fonts of the original IBM and Tandy adapters (which are in the public domain in a number of countries) have been included in PC-BASIC. These provide the most accurate emulation. However, the font ROMs only included a single code page – DOS Latin USA 437.

To support code pages beyond 437, PC-BASIC defaults to the **CPIDOS** font used by default in FreeDOS and DOSBox. This font is very similar in style to the IBM VGA font but has support for many more code pages, in particular Western and Middle Eastern alphabets. If CPIDOS does not include the characters you need, PC-BASIC tries to find them in **UNI-VGA**, which is also very similar in style to VGA. If that does not suffice, PC-BASIC falls back to **GNU UniFont** for the missing characters. UniFont's Western glyphs are less similar to the VGA font, but it provides support for the full Unicode Basic Multilingual Plane. For example, all Chinese, Japanese and Korean glyphs in PC-BASIC are taken from UniFont.

It is possible to change the choice of font using the `font=` option. You can provide a list of fonts, where the last font specified is the most preferred one.

PC-BASIC reads fonts in the `.hex` format introduced by UniFont. It's easy to define custom fonts in this format: it can be edited in a regular text editor. See the UniFont project for an authoring tool. You can add custom fonts to PC-BASIC by installing them into the `font/` subdirectory of PC-BASIC's installation directory.

By default, the following fonts are available:

`font_name`	Name	Sizes	Codepages
unifont	GNU UniFont	16	all
univga	Uni-VGA font	16	many
freedos	FreeDOS CPIDOS font	8, 14, 16	many
cga	IBM Colour/Graphics Adapter font	8	437 only
mda	IBM Monochrome Display Adapter font	14	437 only
vga	IBM Video Graphics Array font	8, 14, 16	437 only
olivetti	Olivetti/AT&T font	16	437 only
tandy1	Tandy-1000 font old version	8	437 only
tandy2	Tandy-1000 font new version	8	437 only

GNU UniFont contains glyphs for all the defined code points of the Basic Multilingual Plane of Unicode 7.0, which makes it an excellent fallback font. Its glyphs are not, however, very similar to those that were used on DOS machines; both Uni-VGA and the FreeDOS CPIDOS fonts are therefore more suitable, when they have the required glyphs.

Redirecting I/O

PC-BASIC supports redirecting input and output the GW-BASIC way: output redirected with the `output=` option will be sent to the screen as well as the specified file, while input redirected with `input=` is taken only from the specified file. Note that screen output through the `SCRN:` device and keyboard input through the `KYBD:` device are not redirected. Files are read and written in the codepage set with PC-BASIC.

Note that it is also possible to use your operating system's facility to redirect console output using the `<` and `>` operators. It's best to set `interface=none` so that I/O is redirected through the console. This will produce files in your console's standard encoding, which is often UTF-8 on Unix and Windows-1252 on Windows.

Command-line interface

You can run PC-BASIC as a command-line interface by setting the `interface=cli` (or `-b`) option. No window will be opened: you can type BASIC commands straight into your Command Prompt/Terminal. Use the horizontal arrow keys to move on the current line you're editing; use the vertical arrow keys to show screen rows above and below. Copy and paste are available only if the calling shell provides them. On Windows, `Ctrl`+`Break` will terminate PC-BASIC immediately. You can use `Ctrl`+`C` to interrupt the program. The end-of-file key combination (`Ctrl`+`D` on Unix, `Ctrl`+`Z` on Windows) will exit PC-BASIC.

You can use the command-line interface to run one or a few BASIC commands directly, like so:

```
me@mybox$ pcbasic -bqe="?1+1" 2 me@mybox$
```

For scripting purposes, it is also possible to run PC-BASIC without any interface by setting `interface=none` or `-n` . If this is set, PC-BASIC will take input from and send output to the console as UTF-8 without further modification. This is useful in combination with redirection and pipes.

Text-based interface

There is also a full-screen text interface available: enable it by setting `interface=text` (or `-t`). The text-based interface is very similar to the default graphical interface, but runs in your Command Prompt or Terminal window.

Graphical screen modes can be used in text and command-line interface, but only the text on the screen will be visible. pre, many `Ctrl` and `Alt` key combinations are not available.

The text and command-line interfaces will attempt to use the PC speaker for sound. Only single-voice sound can be produced this way. On Linux systems under X11, you may need to install the `beep` utility and enable the PC-speaker driver or emulation; direct speaker access is often limited to root or tty logins, and on Ubuntu systems it is disabled by default.

4. Configuration guide

This documentation discusses how to change settings and options for PC-BASIC.

4.1. Changing settings

PC-BASIC has a number of settings that change the way it operates. Settings can be changed by setting options on the command line or through editing the configuration file. In either method, the options have the same name and syntax. In what follows, we will often refer to a particular option setting; remember that you can set this from the command line as well as from the configuration file.

Command-line options

You can enter command-line options if you start PC-BASIC from your operating system's command prompt, console or terminal (the `C:\>` prompt on Windows), by supplying the option with two dashes in front, like so:

```
pcbasic --preset=tandy --ctrl-c-break=True
```

On the command line, you can leave out the expression `=True` that is common in switching options. Some options have an alternative, short name consisting of a single letter preceded by a single dash, which you can use on the command line. You can combine multiple short options with a single dash.

Configuration files

You can change options by adding or removing lines in your local configuration file, which can be found in the following location:

Windows

```
%AppData%\pcbasic-2.0\PCBASIC.INI
```

OS X

```
~/Library/Application Support/pcbasic-2.0/PCBASIC.INI
```

Linux

```
~/.config/pcbasic-2.0/PCBASIC.INI
```

Change an option in the configuration file by adding a line in the section named `[pcbasic]`, like so:

```
[pcbasic]
preset=tandy
ctrl-c-break=True
```

You cannot use <u>positional arguments</u> or the short name of options in the configuration file. You also cannot leave out the expression `=True` .

The configuration file should be a text file encoded in ASCII or UTF-8.

4.2. Synopsis

```
pcbasic [program|package [output]] [--allow-code-poke[=True|=False]] [--aspect=x,y] [-b]
    [--border=width] [--caption=title] [--cas1=type:value]
    [--codepage=codepage_id[:nobox]] [--config=config_file] [--com1=type:value]
    [--com2=type:value] [--convert={A|B|P}] [--mouse-clipboard[=True|=False]]
    [--ctrl-c-break[=True|=False]] [--current-device={CAS1|@|A|B ... |Z}]
    [--debug[=True|=False]] [--dimensions=x,y] [-d] [--double[=True|=False]]
    [-e=statement[:statement ...]] [--exec=statement[:statement ...]]
    [--extension=module_name[,module_name ... ]] [--font=font_name[,font_name ... ]]
    [--fullscreen[=True|=False]] [-h] [--help] [--hide-listing=line_number]
    [--hide-protected[=True|=False]] [--input=input_file]
    [--interface=[none|cli|text|graphical]] [-k=keystring] [--keys=keystring]
    [-l=program] [--load=program] [--logfile=log_file] [--lpt1=type:value]
    [--lpt2=type:value] [--lpt3=type:value] [-f=number_of_files]
    [--max-files=number_of_files] [--max-memory=max_memory[,basic_memory_blocks]]
    [-s=record_length] [--max-reclen=record_length]
    [--monitor={rgb|composite|green|amber|grey|mono}]
    [--mount=drive:path,[drive:path ... ]] [-n] [--output=output_file[:append]]
    [--peek=[seg:addr:val[,seg:addr:val ... ]]] [--preset=option_block]
    [--prevent-close[=True|=False]] [-q] [--quit[=True|=False]]
    [--reserved-memory=number_of_bytes] [--resume[=True|=False] [-r=program]
    [--run=program] [--scaling={smooth|crisp|native}] [--serial-buffer-size=size]
    [--shell=[shell-executable]] [--soft-linefeed[=True|=False]]
    [--sound=[none|beep|portaudio|interface]] [--state=state_file]
    [--syntax={advanced|pcjr|tandy}] [-t] [--term=terminal_program]
    [--text-width={40|80}] [--utf8[=True|=False]] [-v] [--version]
    [--video=adapter[:low]] [--video-memory=size] [-w] [--wait[=True|=False]]
    [--options=gwbasic_options]
```

4.3. Positional arguments

Positional arguments must come before any options, must not start with a dash `-` . Any positional arguments that follow options will be ignored.

program

> If a `.BAS` program is specified as the first positional argument, it will be run. The `--run` , `--load` and `--convert` options override this behaviour.

package

> If a zipfile package or directory is specified as the first positional argument, any contained configuration file `PCBASIC.INI` will be loaded; usually, it will run a program file in the package. All other command-line options will override the package configuration file, note in particular the potential of the `--run` , `--load` and `--convert` options to alter the behaviour of the package.

output

> If a second positional argument is specified, it sets the output file for file format conversion. This argument is ignored unless the `--convert` option is given.

4.4. Options

`--allow-code-poke``[=True|=False]`

> Allow programs to `POKE` into code memory.

`--aspect=``x,y`

> Set the display aspect ratio to `x` : `y` . Only has an effect if combined with `--interface=graphical` .

`-b`

> Use the command-line interface. This is identical to `--interface=cli` .

`--border=``width`

> Set the width of the screen border as a percentage from 0—100. The percentage refers to the total width of the borders on both sides as a fraction of the usable screen width. Only has an effect if combined with `--interface=graphical` .

`--caption=``title`

> Set the title bar caption of the PC-BASIC window. Default `title` is *PC-BASIC*.

`--cas1=``type:value`

> Attach a resource to the `CAS1:` cassette device. `type:value` can be

> **`WAV:`**`wav_file`

>> Connect to the RIFF Wave file `wav_file` with data modulated in IBM PC cassette format.

> **`CAS:`**`cas_file`

>> Connect to the PCE/PC-BASIC CAS tape image `cas_file` .

`--codepage=``codepage_id`**`[:nobox]`**

> Load the specified codepage. The codepage determines which characters are associated to a given character byte or, in the case of double-byte codepages, two character bytes. The available codepages are stored in the `codepage/` directory; by default, these are:

`437`	`720`	`737`	`775`	`806`	`850`
`851`	`852`	`853`	`855`	`856`	`857`
`858`	`860`	`861`	`862`	`863`	`864`
`865`	`866`	`868`	`869`	`874`	`932`
`934`	`936`	`938`	`949`	`950`	`1258`

				georgian-academy	georgian-ps
`alternativ`	`armscii8a`	`big5-2003`	`big5-hkscs`		
`iransystem`	`iscii-as`	`iscii-be`	`iscii-de`	`iscii-gu`	`iscii-ka`
`iscii-ma`	`iscii-or`	`iscii-pa`	`iscii-ta`	`iscii-te`	`kamenicky`
`koi8-r`	`koi8-ru`	`koi8-u`	`mazovia`	`mik`	`osnovnoj`
`pascii`	`ruscii`	`russup3`	`russup4ac`	`russup4na`	`viscii`

. See the list of codepages in the User's Guide for details.

The specifier `nobox` disables box-drawing recognition for double-byte character set code pages. By default, sequences of box-drawing characters are recognised by an algorithm that isn't as smart as it thinks it is, and displayed as box drawing rather than as DBCS characters. If `nobox` is set, they will be displayed as DBCS.

`--config=`*config_file*

Read a configuration file. The system default configuration is always read first, but any preset group of options in a configuration file replaces the whole equivalent default preset group.

`--com1=`*type:value*

Attach a resource to the `COM1:` serial device. *type:value* can be one of the following.

`PORT:`*device_name*

Connect to a serial device. *device_name* can be a device name such as `COM1` or `/dev/ttyS0` or a number, where the first serial port is number 0.

`SOCKET:`*host:socket*

Connect to a TCP socket on a remote or local host.

`RFC2217:`*host:socket*

Connect using the RFC2217 protocol to a TCP socket on a remote or local host.

`STDIO:`[`CRLF`]

Connect to standard I/O of the calling shell. If `CRLF` is specified, PC-BASIC replaces `CR` characters with `LF` on its output and `LF` with `CR` on its input. This is more intuitive on Unix shells. When using a Unix console, you should use `stty -icanon` to enable PC-BASIC to read input correctly.

If this option is not specified, the `COM1:` device is unavailable.

`--com2=`*type:value*

Attach a resource to the `COM2:` serial device. See `--com1` .

`--convert={A|B|P}`

Convert program to one of the following formats:

`A`	Plain text
`B`	Tokenised
`P`	Protected

If `output` is not specified, write to standard output. If program is not specified, use the argument of `--run` or `--load`. If none of those are given, read from standard input. Overrides `--resume`, `--run` and `--load`.

`--mouse-clipboard[=True|=False]`

Enable clipboard operations with the mouse. If `True` (default), select text with the left mouse button to copy and paste with the middle mouse button.

`--ctrl-c-break[=True|=False]`

If `False`, follow GW-BASIC behaviour where `Ctrl`+`C` breaks `AUTO` and `INPUT` but not program execution or `LIST`.
If `True`, treat `Ctrl`+`C` exactly like `Ctrl`+`Break` and `Ctrl`+`Scroll Lock` when `--interface=graphical`.
With `--interface={text|cli}`, `Ctrl`+`C` is always treated like `Ctrl`+`Break`.
Default is `True`.

`--current-device={CAS1|@|A|B ... |Z}`

Set the current device to the indicated PC-BASIC drive letter or `CAS1` for the cassette device. The device chosen should be mounted to an actual location using `--mount` (or `--cas1` if the cassette device is chosen).

`--debug[=True|=False]`

Developer option - use only if you know what you're doing.
Enable debugging extension.

`--dimensions=x,y`

Set window dimensions to `x` by `y` pixels. This overrides `--scaling=native` and `--aspect`. Only has an effect if combined with `--interface=graphical`.

`-d` `--double[=True|=False]`

Enable double-precision transcendental math functions. This is equivalent to the `/d` option in GW-BASIC.

`-e=statement[:statement ...]` `--exec=statement[:statement ...]`

Execute BASIC statements. The `statement`s are executed after loading any program but before entering into direct mode or running it. Multiple statements can be entered

by separating them with colons `:` . These will be executed as if they were entered as separate statements, not as a single compound statement; even if statements such as `GOTO` can be included, the following statements will be executed. If a `:` is intended to be part of a statement, the statement must be quoted with single quotes `'` . If your calling shell interprets such quotes, you should properly escape them.

`--extension=`*module_name*`[,`*module_name ... *`]`
Developer option - use only if you know what you're doing.
Load extension module(s).

`--font=`*font_name*`[,`*font_name ... *`]`
Use the specified fonts for the interface. The last fonts specified take precedence, previous ones are fallback. Default is `unifont,univga,freedos` (i.e. the `freedos` font has preference). The available fonts are stored in `font/` . By default, the following fonts are available:

| `unifont` | `univga` | `freedos` | `cga` | `mda` | `vga` |
| `olivetti` | `tandy1` | `tandy2` | | | |

. See the list of fonts in the User's Guide for details.

`--fullscreen`[`=True`|`=False`]
Fullscreen mode. Only has an effect if combined with `--interface=graphical` .

`-h` `--help`
Show a usage message and exit.

`--hide-listing=`*line_number*
Disable listing and saving to plain text of lines beyond *line_number* , as in GW-BASIC beyond `65530` . Use with care as this allows execution of hidden lines of code. Default is to list all lines.

`--hide-protected`[`=True`|`=False`]
Disable listing and saving to plain text of protected files, as in GW-BASIC. Use with care as this allows execution of hidden lines of code.

`--input=`*input_file*
Retrieve keyboard input from *input_file* , except if `KYBD:` is read explicitly. Input from `KYBD:` files is always read from the keyboard, following GW-BASIC behaviour. If *input_file* is `STDIO:` or `STDIN:` , keyboard input will be read from standard input.

`--interface=`[`none`|`cli`|`text`|`graphical`]
Choose the type of interface. Not all interfaces will be available on all systems. The following interface types may be available:

`none`	Filter for use with pipes. Also `-n` .
`cli`	Command-line interface. Also `-b` .
`text`	Text interface. Also `-t` .
`graphical`	Graphical interface.
`sdl2`	SDL2 graphical interface.
`pygame`	PyGame graphical interface.
`ansi`	ANSI text interface.
`curses`	NCurses text interface.

The default is `graphical` .

`-k=`*keystring* `--keys=`*keystring*

Insert the `keystring` into the keyboard buffer. `keystring` may contain escape codes such as `\r` for return, `\n` for line feed and `\x`*XX* to enter `CHR$(&H`*XX*`)` . `keystring` may contain <u>e-ASCII codes</u> to indicate keypresses that do not have a regular character encoding. For example, `\0\x0F` indicates Shift+Tab.

`-l=`*program* `--load=`*program*

Start in direct mode with the BASIC `program` loaded.

`--logfile=`*log_file*

Write error and warning messages to `log_file` instead of `stderr` .

`--lpt1=`*type*`:`*value*

Attach a resource to the `LPT1:` parallel device. `type:value` can be

`PRINTER:[`*printer_name*`][:`*trigger*`]`

Connect to a Windows, LPR or CUPS printer. If `printer_name` is not specified, the default printer is used.
The printer is activated when a file open to it is closed, a program terminates or PC-BASIC exits. Note that, unlike `LPT1:` , printers connected to `LPT2:` or `LPT3:` do *not* get activated when a program terminates. If specified, `trigger` sets an additional trigger to activate the printer:

`line`	After every line break.
`page`	After every page break.
`close`	No additional trigger

The default behaviour is that of `close` .

`FILE:`*file_name*

Connect to any file or device such as `/dev/stdout` on Unix or `LPT1` on

Windows.

`STDIO:`[`CRLF`]

> Connect to standard output of the calling shell. If `CRLF` is specified, PC-BASIC replaces `CR` characters with `LF` on its output. This is more intuitive on Unix shells.

`PARPORT:`*port_number*

> Connect to a Centronics parallel port, where *port_number* is `0` for the first parallel port, etc. `PARPORT` only works with physical parallel ports; for example, a Windows printer or other device mapped with `NET USE LPT1:` can only be attached with `FILE:LPT1`

If this option is not specified, `LPT1:` is connected to the default printer.

`--lpt2=`*type*`:`*value*

> Attach a resource to the `LPT2:` parallel device. See `--lpt1`. If this option is not specified, `LPT2:` is unavailable.

`--lpt3=`*type*`:`*value*

> Attach a resource to the `LPT3:` parallel device. See `--lpt1`. If this option is not specified, `LPT3:` is unavailable.

`-f=`*number_of_files*　`--max-files=`*number_of_files*

> Set maximum number of open files to *number_of_files*. This is equivalent to the `/f` option in GW-BASIC. Default is `3`.

`--max-memory=`*max_memory*[`,`*basic_memory_blocks*]

> Set the maximum size of the data memory segment to *max_memory* and the maximum size of the data memory available to BASIC to *basic_memory_blocks*`*16`. In PC-BASIC, the minimum of these values is simply the data memory size; the two values are allowed for compatibility with the `/m` option in GW-BASIC.

`-s=`*record_length*　`--max-reclen=`*record_length*

> Set maximum record length for `RANDOM` files to *record_length*. Default is `128`, maximum is `32767`. This is equivalent to the `/s` option in GW-BASIC.

`--monitor=`{`rgb`|`composite`|`green`|`amber`|`grey`|`mono`}

> Sets the monitor type to emulate. Available types are:

`rgb`	RGB colour monitor (default).
`composite`	Composite colour monitor.
`green`	Green-tinted monochrome monitor.
`amber`	Amber-tinted monochrome monitor.
`grey`	Greyscale monochrome monitor.
`mono`	Green-tinted monochrome monitor (same as `green`).

On `SCREEN 2` with `--video={pcjr|tandy|cga|cga_old}` , `--monitor=composite` enables (crude) colour artifacts.

`--mount=`*drive*`:`*path*`,[`*drive*`:`*path* `... ]`

Assign the path `path` to drive letter `drive:` . The path can be absolute or relative. If this option is not specified: on Windows, all Windows drive letters will be assigned to PC-BASIC drive letters; on other systems, the current working directory is assigned to `Z:` .

`-n`

Run PC-BASIC as a command-line filter. Same as `--interface=none` .

`--output=`*output_file*`[:`**append**`]`

Send screen output to `output_file` , except if `SCRN:` is written to explicitly. Output to `SCRN:` files will always be shown on the screen, as in GW-BASIC. If the specifier `append` is given, the output file is appended to rather than overwritten. If `output_file` is `STDIO:` or `STDOUT:` , screen output will be sent to standard output.

`--peek=[`*seg*`:`*addr*`:`*val*`[,`*seg*`:`*addr*`:`*val* `... ]]`

Define `PEEK` preset values. If defined, `DEF SEG` *seg*`:? PEEK(`*addr*`)` will return `val` .

`--preset=`*option_block*

Load machine preset options. A preset option corresponds to a section defined in a config file by a name between square brackets, like

```
[this]
```

`--preset=this` will load all settings defined in that section. Available presets depend on your configuration file. See the list of default presets in the User's Guide.

`--prevent-close[=`**True**`|=`**False**`]`

Suppress window close event. This allows BASIC to capture key combinations that normally close the window. Graphical interface only. By default, the operating system's key combination to close a window (usually `Alt`+`F4`) terminates PC-BASIC. Set `--prevent-close` to allow BASIC to capture this key combination instead. This is useful if your program uses this key combination.

`-q` **`--quit`**[**`=True`**|**`=False`**]

> Quit interpreter when execution stops. If combined with `--run`, PC-BASIC quits when the program ends. If set in direct mode, PC-BASIC quits after the first command is executed.

`--reserved-memory=`*`number_of_bytes`*

> Reserve *`number_of_bytes`* of memory at the bottom of the data segment. For compatibility with GW-BASIC. Default is `3429` bytes. Lowering this value makes more string and variable space available for use by programs.

`--resume`[**`=True`**|**`=False`**

> Resume from saved state. Overrides `--run` and `--load`.

`-r=`*`program`* **`--run=`***`program`*

> Run the specified *`program`*. Overrides `--load`.

`--scaling=`{**`smooth`**|**`crisp`**|**`native`**}

> Choose scaling method.
>
> **`smooth`**
>
> > The display is smoothly scaled to the largest size that allows for the correct aspect ratio.
>
> **`crisp`**
>
> > The display is scaled to the same size as with `smooth`, but without smoothing.
>
> **`native`**
>
> > Scaling and aspect ratio are optimised for the display's native pixel size, without smoothing. **`--scaling=native`** overrides `--aspect.`
>
> Default is `smooth`. Only has an effect if combined with `--interface=graphical`.

`--serial-buffer-size=`*`size`*

> Set serial input buffer *`size`*. Default is `256`. If set to `0`, serial communications are disabled.

`--shell=`[*`shell-executable`*]

> Enable the `SHELL` statement to run the operating system command interpreter *`shell-executable`*. The executable *`shell-executable`* should support MS-DOS's `COMMAND.COM` calling conventions, in particular its `/C` switch. Example command interpreters are `CMD.EXE` on Windows and `"wine cmd.exe"` on Unix. If *`shell-executable`* is empty (as it is by default), the `SHELL` statement is disabled.

`--soft-linefeed`[`=True`|`=False`]

> Do not treat `LF` in text and program files as a line break. This enables the highest level of compatibility with GW-BASIC files. If this option is set, any Linux or Mac text files need to be converted to DOS text before using then with PC-BASIC.

`--sound=`[`none`|`beep`|`portaudio`|`interface`]

> Choose the sound engine to use. Not all sound engines will be available on all systems.

> | | |
> |---|---|
> | `none` | Suppress sound output. |
> | `beep` | Use the built-in speaker. |
> | `portaudio` | Use the PortAudio sound generator. |
> | `interface` | Use the native sound engine of the interface, if available. |

> Default is `interface`.

`--state=`*state_file*

> Set the save-state file to *state_file*. Default is `pcbasic.session` in the Application Data directory.

`--syntax=`{`advanced`|`pcjr`|`tandy`}

> Choose BASIC dialect. Available dialects are:

> | | |
> |---|---|
> | `advanced` | Microsoft GW-BASIC and IBM BASICA |
> | `pcjr` | IBM PCjr Cartridge BASIC |
> | `tandy` | Tandy 1000 GW-BASIC. |

> Default is `advanced`.

`-t`

> Use text-based interface. Same as `--interface=text`.

`--term=`*terminal_program*

> Set the terminal program run by the PCjr `TERM` command to *terminal_program*. This only has an effect with `--syntax=`{`pcjr`|`tandy`}.

`--text-width=`{`40`|`80`}

> Set the number of columns in text mode at startup. Default is `80`.

`--utf8`[`=True`|`=False`]

> Use UTF-8 for plain-text programs and redirected I/O. If `True`, programs stored as plain text will be converted from UTF-8 on `LOAD` and to UTF-8 on `SAVE`. If `False`, the PC-BASIC codepage will be used.

`-v` `--version`

> Print PC-BASIC version string and exit.

`--video=`*adapter*[`:low`]

> Set the video adapter to emulate. Available adapters:

`vga`	Video Graphics Array
`ega`	Enhanced Graphics Adapter
`cga`	Color/Graphics Adapter
`cga_old`	Color/Graphics Adapter (old version; affects composite)
`mda`	Monochrome Display Adapter
`hercules`	Hercules Graphics Adapter
`pcjr`	IBM PCjr graphics
`tandy`	Tandy 1000 graphics
`olivetti`	Olivetti M24 graphics

> Default is `vga` .
>
> If the specifier `low` is given, use low-intensity palettes in CGA modes. This only has an effect with adapters `{cga|cga_old|ega|vga}` .

`--video-memory=`*size*

> Set the amount of emulated video memory available. This affects the number of video pages that can be used. On PCjr and Tandy, this can be changed at runtime through the `CLEAR` statement; at least 32768 needs to be available to enter `SCREEN 5` and `SCREEN 6` . Default is `16384` or PCjr and Tandy and `262144` on other machine presets.

`-w` `--wait`[`=True`|`=False`]

> If `True` , PC-BASIC waits for a keystroke before closing the window on exit. Only works for `--interface=graphical` or `--interface=text` . Default is `False` .

`--options=`*gwbasic_options*

> Set GW-BASIC-style command-line switches. This is a convenience option to facilitate migration from GW-BASIC. `gwbasic_options` is a string that may contain the following options:
>
> `/d`
>
> > Enable double-precision floating-point math functions. See also `--double` .
>
> `/f:`*n*
>
> > Set the maximum number of open files. See also `--max-files` .

`/s:`*n*

   Set the maximum record length for `RANDOM` files. See also `--max-reclen` .

`/c:`*n*

   Set the size of the receive buffer for `COM` devices. See also `--serial-buffer-size` .

`/i`

   Statically allocate file control blocks and data buffer. Note that this is already the default approach in GW-BASIC and PC-BASIC so that this option has no effect.

`/m:`*n,m*

   Set the highest memory location to `n` and maximum BASIC memory size to `m*16` bytes. See also `--max-memory` .

`>`*filename*

   Write screen output to `filename` . See also `--output` .

`>>`*filename*

   Append screen output to `filename` . See also `--output` .

`<`*filename*

   Read keyboard input from `filename` . See also `--input` .

GW-BASIC-style switches are not case sensitive. Note that the symbols used in these switches may have different meaning in the shell from which PC-BASIC is called; you should quote and escape the options as necessary.

4.5. Examples

`pcbasic`

Start PC-BASIC in direct mode, emulating GW-BASIC/BASICA with VGA graphics.

`pcbasic --codepage=950`

Start PC-BASIC using the Big-5 codepage.

`pcbasic Foobar.baz`

Start PC-BASIC with package Foobar. Load the settings from the package; usually this will run a main program contained in the package.

`pcbasic Foobar.baz --convert=A --utf8`

List the main program of package Foobar to standard output as UTF-8.

`pcbasic MYPROG.BAS --mount=A:./files,B:./morefiles`

Mount the current directory's subdirectory `files` as drive `A:` and subdirectory `morefiles` as drive `B:`, then run `MYPROG.BAS`.

`pcbasic --mount=A:C:\fakeflop`

Run PC-BASIC with Windows directory `C:\fakeflop` mounted as `A:` drive.

`pcbasic Z:\INFO.BAS --preset=mda --monitor=amber`

Run `INFO.BAS` in the current directory on an emulated MDA with amber tinted monitor.

`pcbasic /home/me/retro/COMP.BAS --preset=cga --monitor=composite`

Run `COMP.BAS` stored at `/home/me/retro` on an emulated CGA machine with a composite monitor.

`pcbasic PCJRGAME.BAS --preset=pcjr -k='start\r'`

Run `PCJRGAME.BAS` on an emulated PCjr and feed in the keystrokes `s` `t` `a` `r` `t` `Enter`.

`pcbasic BANNER.BAS --lpt2=PRINTER:`

Run `BANNER.BAS` in default mode with the default printer attached to `LPT2:`.

`pcbasic --resume`

Resume the most recently closed PC-BASIC session.

`pcbasic -qbe="?1+1"`

> Execute the BASIC command `PRINT 1+1` in the command-line interface and return to the calling shell.

5. Language guide

This documentation describes the PC-BASIC language, which aims to faithfully emulate GW-BASIC 3.23, IBM Advanced BASIC, IBM Cartridge BASIC and Tandy 1000 GW-BASIC.

The BASIC Language Guide covers the language topic by topic, thematically grouping language elements used for a related purpose. Please refer to the BASIC Language Reference for a formal description of the langage elements and their syntax.

5.1. Working with programs

Statement	Description
AUTO	Enter automatic line numbering mode
CHAIN	Load a new program and run it, preserving common variables
COMMON	Set common variables
DELETE	Delete lines from the program
EDIT	Print a program line to the screen for editing
LIST	Print program lines to the screen
LLIST	Print program lines to the printer
LOAD	Read a new program from file
MERGE	Overlay a program file onto the current program
NEW	Clear the current program from memory
RENUM	Replace the program's line numbers
RUN	Start the current program
SAVE	Store the current program to file
TRON	Enable line number tracing
TROFF	Disable line number tracing
SYSTEM	Exit the BASIC interpreter

5.2. Control flow

A program is normally executed starting with its lowest line number (or the line number called by `RUN`). Statements on a line are executed from left to right. When all statements on a line are finished, execution moves to the next lowest line number, and so on until no line numbers are left. Control flow statements can be used to modify this normal flow of executon.

The `END` and `STOP` statements serve in a program to stop its execution and return to direct mode. When `STOP` is used, a `Break` message is printed. From direct mode, `CONT` can be executed to resume the program where it was stopped. While `END` seems intended to terminate the program, it does not preclude the user from resuming it with `CONT` .

Unconditional jumps can be made with `GOTO` . The program flow will continue at the line number indicated in the `GOTO` statement. Due to the PC-BASIC language's lack of sophisticated looping, branching and breaking constructs, unconditional jumps are essential and used frequently.

The `GOSUB` statement jumps to a subroutine. Similar to `GOTO` , this is an unconditional jump; however, the location of the call is stored and the program will continue its flow there after the subroutine terminates with a `RETURN` statement. Subroutines are somewhat like procedures in that they allow chunks of code that perform a given task to be separated from the main body of the program, but they do not have separate scope since all variables in PC-BASIC are global. They do not have return values. It is even possible to jump out of a subroutine to anywhere in the program by supplying the `RETURN` statement with a line number.

The `ON` statement provides an alternative branching construct. An integer value is used to selects one of a list of line numbers, and execution is continued from there. It can be used with a `GOTO` jump as wellas with a `GOSUB` subroutine call.

`ON` , `GOTO` and `GOSUB` can also be used from direct mode to start a program or subroutine without resetting variables.

The `IF-THEN-ELSE` construct tests for a condition and execute different code branches based on its truth value. This is not a block construct; all code in the `THEN` and `ELSE` branches must fit on one line. For this reason, branching is often used in combination with `GOTO` jumps. For example:

```
10 INPUT "How old are you"; AGE%
20 IF AGE%>30 THEN 100
30 IF AGE%<30 THEN 200 ELSE PRINT "You are 30 years old."
40 END
100 PRINT "You are over 30."
110 END
200 PRINT "You are not yet 30."
210 END
```

The `WHILE-WEND` looping construct repeats the block of code between `WHILE` and `WEND` as long as a given condition remains true.

The `FOR-NEXT` construct repeats a block of code while a counter remains in a given range. The counter is set to a starting value at the first pass of the `FOR` statement and incremented by the `STEP` value at each pass of `NEXT`. For example:

```
10 FOR I=1 TO 10
20 PRINT STRING$(I, "*"); USING " [##]"; I
30 NEXT I
```

Looping constructs may be nested.

Control flow is also affected by event and error trapping.

Statement	Description
CONT	Continue interrupted program
ELSE	Ignore the remainder of the line (standalone ELSE)
END	Stop execution of the program
FOR	Start a for-loop
GOSUB	Call a subroutine
GOTO	Jump to another location in the program
IF	Branch on a condition
NEXT	Iterate a for-loop
ON	Calculated jump or subroutine call
RETURN	Return from subroutine
STOP	Interrupt program execution
WEND	Iterate a while-loop
WHILE	Enter a while-loop

5.3. Arrays and variables

Statement	Description
DEFDBL	Specify variable name range for double-precision floats
DEFINT	Specify variable name range for integers
DEFSNG	Specify variable name range for single-precision floats
DEFSTR	Specify variable name range for strings
DIM	Allocate an array
ERASE	Deallocate an array
LET	Assign a value to a variable
OPTION BASE	Set the starting index of arrays
SWAP	Swap two variables

5.4. Type conversion

Function	Description
ASC	Character to ordinal value
CHR$	Ordinal value to character
HEX$	Integer to hexadecimal string representation
OCT$	Integer to octal string representation
STR$	Numeric value to decimal string representation
VAL	String representation to numeric value
CDBL	Numeric value to double-precision float
CINT	Numeric value to integer
CSNG	Numeric value to single-precision float
CVD	Byte representation to double-precision float
CVI	Byte representation to integer
CVS	Byte representation to single-precision float
MKD$	Double-precision float to byte representation
MKI$	Integer to byte representation
MKS$	Single-precision float to byte representation

5.5. String operations

Statement	Description
LSET	Copy a left-justified value into a string buffer
MID$	Copy a value into part of a string buffer
RSET	Copy a right-justified value into a string buffer

Function	Description
INSTR	Find
LEFT$	Left substring
LEN	String length
MID$	Substring
RIGHT$	Right substring
SPACE$	Repeat spaces
STRING$	Repeat characters

5.6. Text and the screen

Statement	Description
CLS	Clear the screen
COLOR	Set colour and palette values
LOCATE	Set the position and shape of the text screen cursor
PALETTE	Assign a colour to an attribute
PALETTE USING	Assign an array of colours to attributes
PCOPY	Copy a screen page
PRINT	Print expressions to the screen
VIEW PRINT	Set the text scrolling region
WIDTH	Set the number of text columns on the screen
Function	**Description**
CSRLIN	Current row of cursor
POS	Current column of cursor
SCREEN	Character or attribute at given location

5.7. The printer

Statement	Description
LCOPY	Do nothing
LPRINT	Print expressions to the printer
Function	**Description**
LPOS	Column position of printer head

5.8. Keyboard input

Statement	Description
INPUT	Retrieve user input on the console
LINE INPUT	Retrieve a line of user input on the console
Function	**Description**
INKEY$	Nonblocking read from keyboard
INPUT$	Blocking read from keyboard

5.9. Function-key macros

Statement	Description
KEY	Manage the visibility of the function-key macro list
KEY	Define a function-key macro

5.10. Calculations and maths

Mathematical functions

Function	Description
ABS	Absolute value
ATN	Arctangent
COS	Cosine
EXP	Exponential
FIX	Truncation
INT	Floor
LOG	Natural logarithm
SIN	Sine
SGN	Sign
SQR	Square root
TAN	Tangent

Random numbers

Statement	Description
RANDOMIZE	Seed the random number generator

Function	Description
RND	Pseudorandom number

5.11. Devices and files

File operations

Statement	Description
CLOSE	Close a file
FIELD	Assign a string to a random-access record buffer
GET	Read a record from a random-access file
INPUT	Read a variable from a file
LINE INPUT	Read a line from a file
LOCK	Locks a file or a range of records against other use
OPEN	Open a data file
PUT	Write the random-access record buffer to disk
RESET	Close all files
UNLOCK	Unlocks a file or a range of records against other use
WIDTH	Set the number of text columns in a file
WRITE	Write expressions to a file
Function	**Description**
EOF	End of file
LOC	Location in file
LOF	Length of file
INPUT$	Read a string from a file

Devices

PC-BASIC recognises the following DOS-style devices, which can be used by opening a file on them. Some devices allow specification of further file parameters, such as handshake specifications for serial devices, a filename for cassette devices and a path for disk devices. When operating on disk devices, specifying a path is mandatory.

The *filename aliases* listed here are only available if the current device is a disk device.

Device	Filename alias	Allowed modes	Description
SCRN:	CON	OUTPUT	The screen. Output to SCRN: has largely the same effect as straight output using PRINT. A difference is the WIDTH setting which is independent of the real screen width.
KYBD:	CON	INPUT	The keyboard. Input read from KYBD: is not echoed to the screen. Special keys like arrow keys are registered differently than when using INPUT or INPUT$ straight.
LPT1: LPT2: LPT3:	PRN for LPT1:	OUTPUT RANDOM	Parallel ports 1—3. LPT devices can be attached to the physical parallel port, to a printer or to a text file with the --lpt*n* options. Opening a printer for RANDOM has the same effect as opening it for OUTPUT; attempting random-file operations will raise Bad file mode.
COM1: COM2:	AUX for COM1:	INPUT OUTPUT APPEND RANDOM	Serial ports 1—2. COM devices can be attached to a physical serial port or to a network socket with the --com*n* options.
CAS1:		INPUT OUTPUT	Cassette tape driver. CAS devices can be attached to a WAV (RIFF Wave) or a CAS (bitmap tape image) file with the --cas1 option.
A: — Z: and @:		INPUT OUTPUT APPEND RANDOM	Disk devices. These devices can be mounted to a directory on the host file system with the --mount option.
	NUL	INPUT OUTPUT APPEND RANDOM	Null device. This device produces no bytes when opened for INPUT and absorbs all bytes when opened for OUTPUT.

GW-BASIC additionally recognises the following little-used device, which is not implemented in PC-BASIC.

Device	Allowed modes	Description
`CONS:`	`OUTPUT`	The screen (console). Output to `CONS:` is displayed directly at the cursor position when `Enter` is pressed. It does not update the end-of-line value for the interpreter, which means that it does not move with `Backspace` or `Del` and is not stored in program lines if it appears beyond the end of the existing line. `CONS:` can be opened with any access mode, but the effect is always to open it for `OUTPUT`.

5.12. Graphics

Statement	Description
CIRCLE	Draw an ellipse or arc section
DRAW	Draw a shape defined by a Graphics Macro Language string
GET	Store a screen area as a sprite
LINE	Draw a line segment
PAINT	Flood-fill a connected region
PSET	Put a pixel
PRESET	Change a pixel to background attribute
PUT	Draw a sprite to the screen
SCREEN	Change the video mode
VIEW	Set the graphics viewport
WINDOW	Set logical coordinates
Function	**Description**
POINT	Graphical pointer coordinates
POINT	Pixel attribute
PMAP	Convert between physical and logical coordinates

5.13. Sound

Statement	Description
BEEP	Beep the speaker
BEEP	Speaker switch
NOISE	Generate noise
PLAY	Play a tune encoded in Music Macro Language
SOUND	Generate a tone
SOUND	Sound switch
Function	**Description**
PLAY	Length of the background music queue

5.14. Joystick and pen

Statement	Description
STRIG	Joystick switch
Function	**Description**
PEN	Status of light pen
STICK	Coordinate of joystick axis
STRIG	Status of joystick fire button

5.15. Disks and DOS

The `SHELL` statement is, by default, disabled; this is to avoid unpleasant surprises. In GW-BASIC under MS-DOS, `SHELL` opens a DOS prompt or executes commands in it. The command shells of modern operating systems work differently than those of DOS; in particular, it is impossible to retrieve changes in the environment variables, so that many use cases of `SHELL` simply would not work; for example, changing the current drive on Windows. Moreover, Unix shells have a syntax that is completely different from that of DOS. You can, however, enable `SHELL` by setting the `shell=native` option.

Statement	Description
CHDIR	Change current directory
FILES	List the files in the current directory
KILL	Delete a file on a disk device
MKDIR	Create a new directory
NAME	Rename a file on disk
RMDIR	Remove a directory
ENVIRON	Set a shell environment string
SHELL	Enter a DOS shell
Function	**Description**
ENVIRON$	String from shell environment table

5.16. Serial communications

Statement	Description
GET	Read bytes from a serial port
PUT	Write bytes to a serial port
TERM	Open the terminal emulator

5.17. Event handling

Event trapping allows to define subroutines which are executed outside of the normal course of operation. Events that can be trapped are:

- Time intervals (`ON TIMER`)
- Keypresses (`ON KEY`)
- Serial port input (`ON COM`)
- Music queue exhaustion (`ON PLAY`)
- Joystick triggers (`ON STRIG`)
- Light pen activation (`ON PEN`)

Event trapping subroutines are defined as regular subroutines. At the `RETURN` statement, the normal course of program execution is resumed. Event trapping can be switched on and off or paused temporarily with statements of the form `PEN ON` , `PEN OFF` , `PEN STOP` . Event trapping only takes place during program execution and is paused while the program is in an error trap. If an event occurs while event-trapping is paused, then the event is triggered immediately when event trapping is resumed.

Statement	Description
COM	Manage serial port event trapping
KEY	Manage keyboard event trapping
KEY	Define key to trap in keyboard event trapping
ON	Define event-trapping subroutine
PEN	Manage light pen event trapping
PLAY	Manage music queue event trapping
STRIG	Manage joystick event trapping
TIMER	Manage timer event trapping

PC-BASIC

5.18. Error handling

Normally, any error will interrupt program execution and print a message on the console (exceptions are `Overflow` and `Division by zero`, which print a message but do not interrupt execution). It is possible to handle errors more graciously by setting an error-handling routine with the `ON ERROR GOTO line_number` statement. The error-handling routine starts at the given line number `line_number` and continues until a `RESUME` statement is encountered. Error trapping is in effect both when a program is running and in direct mode. Error trapping is switched off with the `ON ERROR GOTO 0` statement. If an error occurs, or error trapping is switched off, while the program is executing an error-trapping routine, the program terminates and an error message is shown.

Statement	Description
ERROR	Raise an error
ON ERROR	Define an error handler
RESUME	End error handler and return to normal execution
Function	**Description**
ERR	Error number of last error
ERL	Line number of last error

5.19. User-defined functions

Statement	Description
DEF FN	Define a new function

Function	Description
FN	User-defined function

5.20. Date and time

Statement	Description
DATE$	Set the system date
TIME$	Set the system time
Function	**Description**
DATE$	System date as a string
TIME$	System time as a string
TIMER	System time in seconds since midnight

5.21. Including data in a program

Statement	Description
DATA	Define data to be used by the program
READ	Retrieve a data entry
RESTORE	Reset the data pointer

5.22. Memory and machine ports

Only selected memory ranges and selected ports are emulated in PC-BASIC. Some of the most commonly accessed regions of memory are emulated and can be read and (sometimes) written. There is read and write support for video memory, font RAM and selected locations of the low memory segment, including the keyboard buffer. Additionally, there is read support for font ROM, variable, array and string memory, `FIELD` buffers as well as the program code itself. Writing into the program code is disabled by default, but can be enabled with the `allow-code-poke` option. A number of machine ports related to keyboard input and video modes are supported as well.

Statement	Description
BLOAD	Load a binary file into memory
BSAVE	Save a memory region to file
CLEAR	Clears BASIC memory
DEF SEG	Set the memory segment
OUT	Write a byte to a machine port
POKE	Write a byte to a memory location
WAIT	Wait for a value on a machine port
Function	**Description**
FRE	Amount of free memory
INP	Byte at machine port
PEEK	Byte at memory address
VARPTR	Memory address of variable
VARPTR$	Byte representation of length and memory address of variable

5.23. Features not yet implemented

The following language elements are not currently supported in PC-BASIC. The keyword syntax is supported, so no `Syntax error` should be raised if the statements or functions are used correctly. However, the statements do nothing and the functions return zero or the empty string.

These language elements may be implemented in future versions of PC-BASIC.

Statement	Description	PC-BASIC implementation
MOTOR	Turn on cassette motor	Do nothing
Function	**Description**	**PC-BASIC implementation**
ERDEV	Device error value	Return `0`
ERDEV$	Name of device raising error	Return `""`
EXTERR	Extended error information from DOS	Return `0`

5.24. Unsupported features

GW-BASIC was a real-mode DOS program, which means that it had full control over an IBM-compatible 8086 computer. It had direct access to all areas of memory and all devices. Some BASIC programs used this fact, by using machine-code subroutines to perform tasks for which BASIC did not provide support. PC-BASIC runs on modern machines which may be based on completely different architectures and do not allow applications to access the memory directly. Therefore, it is not possible to run machine code on PC-BASIC. If you need machine code, you'll need to use full CPU emulation such as provided by DOSBox, Bochs or VirtualBox.

Similarly, the `IOCTL` functionality depends on an MS-DOS interrupt and sends a *device control string* to any DOS device driver. The syntax of such strings is device-dependent. Since PC-BASIC emulates neither DOS nor whatever device might be parsing the control string, it is not possible to use such functionality.

The following language elements are therefore not supported in PC-BASIC. The keyword syntax is supported, so no `Syntax error` should be raised if the statements or functions are used correctly. However, the statements either do nothing or raise `Illegal function call`; the functions return zero or the empty string or raise `Illegal function call`.

Statement	Description	PC-BASIC implementation
CALL	Call a machine code subroutine	Do nothing
CALLS	Call a machine code subroutine	Do nothing
DEF USR	Define a machine code function	Do nothing
IOCTL	Send a device control string to a device	Raise `Illegal function call`
Function	**Description**	**PC-BASIC implementation**
IOCTL$	Device response to `IOCTL`	Raise `Illegal function call`
USR	Machine code function	Raise `Illegal function call`

6. Language reference

This documentation describes the PC-BASIC language, which aims to faithfully emulate GW-BASIC 3.23, IBM Advanced BASIC, IBM Cartridge BASIC and Tandy 1000 GW-BASIC.

Differences with the original languages do arise, and where this is the case they are documented.

Please note that Microsoft's official documentation for the original languages is rather hit-and-miss; it leaves several features undocumented and incorrectly describes others. To avoid making the same errors, the present documentation was written from scratch with reference to the actual behaviour. The errors in this document are therefore all my own. Please contact me if you encounter them.

6.1. Metasyntax

In descriptions of BASIC syntax, the following conventions apply. Exact rendering of the markup may vary depending on the means used to display this documentation.

`bold`
> Type exactly as shown.

`italic`
> Replace with appropriate metavariable.

`[a]`
> Entities within square brackets are optional.

`{ a | b }`
> Disjunct alternatives of which one must be chosen.

`[ a | b ]`
> Optional disjunct alternatives.

`a ...`
> Preceding entity can be repeated.

6.2. Definitions

A *program line* consists of a line number followed by a compound statement. Program lines are terminated by a `CR` or or by the end of the file (optionally through an `EOF` character). Anything on a program line after a `NUL` character is ignored.

A *line number* is a whole number in the range `[0−65535]`. Note that the line numbers `65530−65535` cannot be entered from the console or a text program file, but can be part of a tokenised program file.

A *compound statement* consists of <u>statements</u> separated by colons:

```
statement [: statement] ...
```

An *expression* takes one of the following forms:

```
unary_operator {literal | variable | array_element | function}
```

```
expression binary_operator expression
```

```
(expression)
```

whose elements are described the sections <u>Literals</u>, <u>Variables</u>, <u>Operators</u> and <u>Functions</u>.

An *array element* takes the form

```
array {[|(} numeric_expression [, numeric_expression ] ... {)|]}
```

6.3. Literals

String literals

String literals are of the following form:

```
"[characters]{NUL|CR|EOF|"}
```

where `characters` is a string of characters. Any character from the current code page can be used, with the following exceptions, all of which terminate the string literal (aside from other effects they may have):

- `NUL` (`CHR$(&h00)`)
- `CR` (`CHR$(&h0D)`)
- `EOF` (`CHR$(&h1A)`)
- `"` (`CHR$(&h22)`)

Strings are also legally terminated by the end of the file in the absence of an `EOF` character.

Apart from these, string literals should not contain any of the characters in the ASCII range `&h0D` — `&h1F` , which lead to unpredictable results. There is no escaping mechanism. To include one of the above characters in a string, use <u>string concatenation</u> and the `CHR$` function.

Numeric literals

Numeric literals have one of the following forms:

```
[+|-] [0|1|2|3|4|5|6|7|8|9]... [.][0|1|2|3|4|5|6|7|8|9]...
    [{E|e|D|d}[+|-][0|1|2|3|4|5|6|7|8|9]...] |#|!|%]
```

```
&{H|h}[0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|a|b|c|d|e|f]...
```

```
&[O|o] [0|1|2|3|4|5|6|7]...
```

Hexadecimal literals must not contain spaces, but decimal and octal literals may. The `o` character in octal literals is optional: they can be specified equally as `&o777` or `&777` .

Hexadecimal and octal literals denote integers and do not include a sign. They must range between [`&h0` — `&hFFFF`], of which the range [`&h8000` — `&hFFFF`] is interpreted as a two's complement negative integer; for example, `&hFFFF = -1` . Signs can appear left of the `&` but these form an expression and are not part of the literal itself.

Floating-point literals must be specified in decimal notation. The decimal separator is the point. A base-10 exponent may be specified after `E` in single-precision floats, or after `D` in double-precision floats. Trailing `%` is ignored and does *not* indicate an integer literal. Trailing `!` or `#` mark the literal as single- or double-precision, respectively.

Examples of valid numeric literals are `-1` `42` `42!` `42#` `1.3523523` `.235435` `-.3` `3.` `.` `.e` `.D` `1.1e+7` `1.1d+7` `1e2` `1e-2` `&7` `&hffff` `&O20` `&h` `&` `65537%` `1.1%`

Note that expressions such as `&o-77` are legal; these are however *not* negative octals but rather the expression `&o` (empty octal; zero) less `77` (decimal 77).

6.4. Variables

Variable names must start with a letter; all characters of the variable name (except the sigil) must be letters `A–Z` , figures `0–9` , or a dot `.` Only the first 40 characters in the name are significant. A variable name must not be identical to a reserved word or a reserved word plus sigil. Therefore, for example, you cannot name a variable `TO!` but you can name it `AS!` . Variable names may *contain* any reserved word. Variable names may also start with a reserved word, with the exception of `USR` and `FN` . Thus, `FNORD%` and `USRNME$` are not legal variable names while `FRECKLE%` and `LUSR$` are.

For each name, four different variables may exist corresponding to the four types. That is, you can have `A$` , `A%` , `A!` and `A#` as different variables. Which one of those is also known as `A` depends on the settings in `DEFINT` / `DEFDBL` / `DEFSNG` / `DEFSTR` . By default, `A` equals the single-precision `A!` .

Furthermore, the arrays `A$()` , `A%()` , `A!()` , `A#()` are separate from the scalar variables of the same name.

Types and sigils

PC-BASIC recognises four variable types, distinguished by their *sigil* or *type character*, the last character of the variable's full name:

sigil	type	size	range	precision
$	string	3 bytes plus allocated string space	0—255 characters	
%	integer	2 bytes	-32768—32767	exact
!	single-precision float	4 bytes	$\pm2.938726\cdot10^{-39}$—$\pm1.701412\cdot10^{38}$	~6 significant figures
#	double-precision float	8 bytes	$\pm2.938735877055719\cdot10^{-39}$—$\pm1.701411834604692\cdot10^{38}$	~16 significant figures

Note that double-precision floats can hold more decimals than single-precision floats, but not larger or smaller numbers.

While all integers are signed, some statements will interpret negative integers as their two's complement.

Arrays

Arrays are indexed with round or square brackets; even mixing brackets is allowed. The following are all legal array elements: `A[0]` , `A(0)` , `A(0]` , `A[0)` . Multidimensional arrays are specified by separating the indices with commas: `A(0, 0)` , `A[0, 0, 0]` , etc.

By default, arrays are indexed from `0` . This can be changed to `1` using `OPTION BASE 1` .

Arrays can be allocated by specifying the largest allowed index using `DIM` . If all indices of the array are `10` or less, they need not be explicitly allocated. The first access of the array (read or write) will automatically allocate it with a maximum index of `10` and the same number of indices as in the first access. To re-allocate an array, the old array must first be deleted with `CLEAR` or `ERASE` .

Multi-dimensional arrays are stored in column-major order, such that `A%(2,0)` immediately follows `A%(1,0)` .

Conversions

PC-BASIC will implicitly convert between the three numerical data types. When a value of one type is assigned to a variable, array element or parameter of another type, it is converted according to the following rules:

- Single- and double-precision floats are converted to integer by rounding to the nearest whole number. Halves are rounded away from zero. If the resulting whole number is outside the allowed range for integers, `Overflow` is raised.
- Double-precision floats are converted to single-precision floats by Gaussian rounding of the mantissa, where the new least significant bit of the mantissa is rounded up if the clipped-off binary fraction is greater than one-half; halves are rounded to even.
- Integers are converted to their exact representation as single- or double-precision floats.
- Single-precision floats are converted to their exact representation as double-precision floats.
- There is no implicit conversion between strings and any of the numeric types. Attempting to assign a string value to a numeric variable, array element or parameter (or vice versa) will raise `Type mismatch` .

6.5. Operators

Order of precedence

The order of precedence of operators is as follows, from tightly bound (high precedence) to loosely bound (low precedence):

12. `^`

    11. `*` `/`

10. `\`

9. `MOD`

    8. `+` `-` (unary and binary)

    7. `=` `<>` `><` `<` `>` `<=` `=<` `>=` `=>`

6. `NOT` (unary)

5. `AND`

4. `OR`

3. `XOR`

2. `EQV`

1. `IMP`

Expressions within parentheses `()` are evaluated first. All binary operators are left-associative: operators of equal precedence are evaluated left to right.

Examples

- Exponentiation is more tightly bound than negation: `-1^2 = -(1^2) = -1` but `(-1)^2 = 1` .
- Exponentiation is left-associative: `2^3^4 = (2^3)^4 = 4096` .

Errors

- If any operator other than `+` , `-` or `NOT` is used without a left operand, `Syntax error` is raised.
- At the end of a statement, if any operator is used without a right operand, `Missing operand` is raised. If this occurs elsewhere inside a statement, such as within brackets, `Syntax error` is raised.

Mathematical operators

Mathematical operators operate on numeric expressions only. Note however that `+` can take the role of the <u>string concatenation operator</u> if both operands are strings.

Code	Operation	Result
`x ^ y`	Exponentiation	x raised to the power of y
`x * y`	Multiplication	Product of x and y
`x / y`	Division	Quotient of x and y
`x \ y`	Truncated division	Integer quotient of x and y
`x MOD y`	Modulo	Integer remainder of x by y (with the sign of x)
`x + y`	Addition	Sum of x and y
`x - y`	Subtraction	Difference of x and y
`+ y`	Unary Plus	Value of y
`- y`	Negation	Negative value of y

Notes

- Where necessary, the result of the operation will be upgraded to a data type able to hold the result. For example, dividing integers 3 by 2 will yield a single-precision 1.5.
- However, the exponentiation operator `^` will give at most a single-precision result unless the <u>`double`</u> option is used.
- The expression `0^0` will return `1` and not raise an error, even though, mathematically, raising zero to the zeroeth power is undefined.

Errors

- If either operand is a string, `Type mismatch` will be raised. The exception is `+` which will only raise `Type mismatch` if either *but not both* operands are strings.
- If `y=0`, `x / y`, `x MOD y` and `x \ y` will raise `Division by zero`.
- If `x=0` and `y<0`, `x^y` will raise `Division by zero`.
- If the result of any operation is too large to fit in a floating-point data type, `Overflow` is raised.
- If operands or result of `\` or `MOD` are not in `[-32768-32767]`, `Overflow` is raised.
- If `x<0` and `y` is a fractional number, `x ^ y` will raise `Illegal function call`.

Relational operators

Relational operators can operate on numeric as well as string operands; however, if one operand is string and the other numeric, `Type mismatch` is raised.

Relational operators return either `0` (for *false*) or `-1` for *true*.

Code	Operation	Result
=	Equal	True if a equals b, false otherwise.
<> ><	Not equal	False if a equals b, true otherwise.
<	Less than	True if a is less than b, false otherwise.
>	Greater than	True if a is greater than b, false otherwise.
<= =<	Less than or equal	False if a is greater than b, true otherwise.
>= =>	Greater than or equal	False if a is less than b, true otherwise.

When operating on numeric operands, both operands are compared as floating-point numbers according to the usual ordering of numbers. The equals operator tests for equality to within machine precision for the highest-precision of the two operator types.

When comparing strings, the ordering is as follows.

- Two strings are equal only if they are of the same length and every character code of the first string agrees with the corresponding character code of the second. This includes any whitespace or unprintable characters.
- Each character position of the strings is compared starting with the leftmost character. When a pair of different characters is encountered, the string with the character of lesser <u>code point</u> is less than the string with the character of greater code point.
- If the strings are of different length, but equal up to the length of the shorter string, then the shorter string is less than the longer string.

Bitwise operators

PC-BASIC has no Boolean type and does not implement Boolean operators. It does, however, implement bitwise operators.

Bitwise operators operate on numeric expressions only. Floating-point operands are rounded to integers before being used.

Code	Operation	Result
NOT *y*	Complement	-*y*-1
x AND *y*	Bitwise conjunction	The bitwise AND of *x* and *y*
x OR *y*	Bitwise disjunction	The bitwise OR of *x* and *y*
x XOR *y*	Bitwise exclusive or	The bitwise XOR of *x* and *y*
x EQV *y*	Bitwise equivalence	NOT(*x* XOR *y*)
x IMP *y*	Bitwise implication	NOT(*x*) OR *y*

These operators can be used as Boolean operators only if `-1` is used to represent *true* while `0` represents *false*. Note that PC-BASIC represents negative integers using the two's complement, so `NOT 0 = -1`. The Boolean interpretation of bitwise operators is given in the table below.

Code	Operation	Result
NOT *y*	Logical negation	True if *y* is false and vice versa
x AND *y*	Conjunction	Only true if both *x* and *y* are true
x OR *y*	Disjunction	Only false if both *x* and *y* are false
x XOR *y*	Exclusive or	True if the truth values of *x* and *y* differ
x EQV *y*	Equivalence	True if the truth values of *x* and *y* are the same
x IMP *y*	Implication	True if *x* is false or *y* is true

Be aware that when used on integers other than `0` and `-1`, bitwise operators can *not* be interpreted as Boolean operators. For example, `2 AND 1` returns `0`.

Errors

- If either operand is a string, `Type mismatch` will be raised.
- If the operands or result are not in `[-32768-32767]`, `Overflow` is raised.

String operators

The string concatenation operator is `+`. It has a binary as well as a unary form. The unary minus may also be used on strings, but has no effect.

Code	Operation	Result
x + *y*	Concatenation	The string formed by *x* followed by *y*
+ *y*	Unary Plus	Value of *y*
- *y*	Unary Minus	Value of *y*

Errors

- If either (but not both) operands to a concatenation are numeric, `Type mismatch` will be raised.
- If `LEN(x) + LEN(y) > 255`, `x + y` will raise `String too long`.

6.6. Functions

Functions can only be used as part of an expression within a statement; they may take input values between parentheses and produce a return value. For example, in `PRINT ABS(-1)` the `ABS` function is used in an expression within a `PRINT` statement; in `Y = SQR(X) + 2` the `SQR` function is used in an expression within a `LET` statement.

Some reference works also use terms such as *system variable* for functions that do not take an input, presumably since in the GW-BASIC syntax such functions have no parentheses, in contrast to the languages in the C family (and indeed some modern BASICs). However, this is simply the GW-BASIC syntax for functions without inputs. For example, one can do `DEF FNA=1: PRINT FNA` in which no parentheses are allowed.

ABS

`y = ABS(x)`

Returns the absolute value of `x` if `x` is a number and the value of `x` if `x` is a string.

Parameters

- `x` is an expression.

ASC

`val = ASC(char)`

Returns the code point (ASCII value) for the first character of `char`.

Parameters

- `char` is an expression with a string value.

Errors

- `char` has a numeric value: `Type mismatch`.
- `char` equals `""`: `Illegal function call`.

ATN

```
y = ATN(x)
```

Returns the inverse tangent of `x` .

Parameters

- `x` is a numeric expression that gives the angle in radians.

Notes

- Unless PC-BASIC is run with the `double` option, this function returns a single-precision value.
- `ATN(x)` differs in the least significant digit from GW-BASIC.

Errors

- `x` has a string value: `Type mismatch` .

CDBL

```
y = CDBL(x)
```

Converts the numeric expression `x` to a double-precision value.

Errors

- `x` has a string value: `Type mismatch` .

CHR$

```
char = CHR$(x)
```

Returns the character with code point `x` .

Parameters

- `x` is a numeric expression in the range `[0–255]` .

Errors

- `x` has a string value: `Type mismatch` .
- `x` is not in `[-32768–32767]` : `Overflow` .
- `x` is not in `[0–255]` : `Illegal function call` .

CINT

```
y = CINT(x)
```

Converts the numeric expression `x` to a signed integer. Halves are rounded away from zero, so that e.g. `CINT(2.5) = 3` and `CINT(-2.5) = -3`.

Errors

- `x` has a string value: `Type mismatch`.
- `x` is not in `[-32768—32767]`: `Overflow`.

COS

```
cosine = COS(angle)
```

Returns the cosine of `angle`. Unless PC-BASIC is run with the `double` option, this function returns a single-precision value.

Parameters

- `angle` is a numeric expression that gives the angle in radians.

Notes

- The return value usually differs from the value returned by GW-BASIC in the least significant figure.

Errors

- `angle` has a string value: `Type mismatch`.

CSNG

```
y = CSNG(x)
```

Converts the numeric expression `x` to a single-precision value by Gaussian rounding.

Errors

- `x` has a string value: `Type mismatch`.

CSRLIN

`y = CSRLIN`

Returns the screen row of the cursor on the active page. The return value is in the range `[1—25]`.

Notes

- This function takes no arguments.

CVI

`y = CVI(s)`

Converts a two-byte string to a signed integer.

Parameters

- `s` is a string expression that represents an integer using little-endian two's complement encoding. Only the first two bytes are used.

Errors

- `s` has a numeric value: `Type mismatch` .

CVS

`y = CVS(s)`

Converts a four-byte string to a single-precision floating-point number.

Parameters

- `s` is a string expression that represents a single-precision number in Microsoft Binary Format. Only the first four bytes are used.

Errors

- `s` has a numeric value: `Type mismatch` .

CVD

```
y = CVD(s)
```

Converts an eight-byte string to a double-precision floating-point number.

Parameters

- `s` is a string expression that represents a double-precision number in Microsoft Binary Format. Only the first eight bytes are used.

Errors

- `s` has a numeric value: `Type mismatch`.

DATE$ (function)

```
s = DATE$
```

Returns the system date as a string in the format `"mm-dd-yyyy"`.

Notes

- This function takes no arguments.

ENVIRON$

```
value = ENVIRON[ ]$(x)
```

Returns an environment variable.

Parameters

`x` is an expression.

- If `x` has a string value, returns the value for the environment variable `x` or the empty string if no variable with the name `x` is set in the environment table.
- If `x` has a numeric value, it must be in `[1–255]`. Returns the `x` th entry in the environment table.

Errors

- `x` is the empty string: `Illegal function call`.
- `x` is a number not in `[-32768–32767]` : `Overflow`.
- `x` is a number not in `[1–255]` : `Illegal function call`.

EOF

```
is_at_end = EOF(file_num)
```

Returns -1 if file with number `file_num` has reached end-of-file; 0 otherwise. The file must be open in `INPUT` or `RANDOM` mode. `EOF(0)` returns 0.

Notes

- If `file_num` is open to `KYBD:`, performs a blocking read and returns -1 if `CTRL`+`Z` is entered, 0 otherwise. The character entered is then echoed to the console.

Errors

- `file_num` has a string value: `Type mismatch`.
- `file_num` is a number not in `[-32768–32767]`: `Overflow`.
- `file_num` is a number not in `[0–255]`: `Illegal function call`.
- `file_num` is not 0 or the number of an open file: `Bad file number`.
- The file with number `file_num` is in OUTPUT or APPEND mode: `Bad file mode`.

ERDEV

```
zero = ERDEV
```

Returns 0.

Notes

- In GW-BASIC, returns the value of a device error.
- This function is not implemented in PC-BASIC.
- This function takes no arguments.

ERDEV$

```
empty = ERDEV[ ]$
```

Returns the empty string.

Notes

- In GW-BASIC, returns the device name of a device error.
- This function is not implemented in PC-BASIC.
- This function takes no arguments.

ERL

```
error_line = ERL
```

Returns the line number where the last error was raised.

Notes

- If the error was raised by a direct statement, returns 65535.
- If no error has been raised, returns 0.
- This function takes no arguments.

ERR

```
error_code = ERR
```

Returns the number of the last error.

Notes

- If no error has been raised, returns 0.
- If the last error was a `Syntax error` raised by a direct statement, returns 0.
- This function takes no arguments.

EXP

```
y = EXP(x)
```

Returns the exponential of `x`, i.e. `e` to the power `x`.

Parameters

- `x` is a number- valued expression.

Notes

- Unless PC-BASIC is run with the `double` option, this function returns a single-precision value.
- The return value sometimes differs in the least significant digit from GW-BASIC. For large values of `x`, the difference may be 3 digits.

Errors

- `x` has a string value: `Type mismatch`.
- `x` is larger than the natural logarithm of the maximum single-precision value: `Overflow`.

EXTERR

`zero = EXTERR(x)`

Returns 0.

Parameters

- `x` is a numeric expression in `[0–3]` .

Notes

- In GW-BASIC, this function returns extended error information from MS-DOS.
- This function is not implemented in PC-BASIC.

Errors

- `x` has a string value: `Type mismatch` .
- `x` is not in `[-32768–32767]` : `Overflow` .
- `x` is not in `[0–3]` : `Illegal function call` .

FIX

`whole = FIX(number)`

Returns `number` truncated towards zero.

Parameters

- `number` is a numeric expression.

Notes

- `FIX` truncates towards zero: it removes the fractional part. By contrast, `INT` truncates towards negative infinity.

Errors

- `number` is a string expression: `Type mismatch` .

FN

```
result = FN[ ]name [(arg_0 [, arg_1] ...)]
```

Evaluates the user-defined function previously defined with `DEF FN` *name* . Spaces between `FN` and *name* are optional.

Parameters

- *name* is the name of a previously defined function.
- *arg_0*, *arg_1*, ... are expressions, given as parameters to the function.

Errors

- No function named *name* is defined: `Undefined user function` .
- The number of parameters differs from the function definition: `Syntax error` .
- The type of one or more parameters differs from the function definition: `Type mismatch` .
- The return type is incompatible with the function name's sigil: `Type mismatch` .
- The function being called is recursive or mutually recursive: `Out of memory` .

FRE

```
free_mem = FRE(x)
```

Returns the available BASIC memory.

Parameters

x is an expression.

- If *x* has a numeric value, it is ignored.
- If *x* has a string value, garbage collection is performed before returning available memory.

HEX$

```
hex_repr = HEX$(x)
```

Returns a string with the hexadecimal representation of `x` .

Parameters

- `x` is a numeric expression in `[-32768–65535]` . Values for negative `x` are shown as two's-complement.

Errors

- `x` is not in `[-32768–65535]` : `Overflow` .
- `x` has a string value: `Type mismatch` .

INKEY$

```
key = INKEY$
```

Returns one key-press from the keyboard buffer. If the keyboard buffer is empty, returns the empty string. Otherwise, the return value is a one- or two- character string holding the <u>e-ASCII code</u> of the pressed key.

Notes

- This function takes no arguments.
- When a function key `F1`–`F10` is pressed, `INKEY$` will return the letters of the associated macro — unless this macro has been set to empty with the `KEY` statement, in which case it returns the e-ASCII code for the function key.

```
code = INP(port)
```

Returns the value of an emulated machine port.

Parameters

port is a numeric expression in `[0–65535]` .

port	Effect
&h60	Returns the keyboard scancode for the current key pressed or the last key released. The scancodes returned by `INP(&h60)` are those listed in the keyboard scancodes table. If a key is currently down, the return value is its scancode. If no key is down, the return value is the scancode of the last key released, incremented by 128.
&h201	Returns the value of the game port (joystick port). This value is constructed as follows: *see table below*

Bit	Meaning
0	joystick 2 x-axis
1	joystick 1 y-axis
2	joystick 1 x-axis
3	joystick 2 y-axis
4	joystick 2 button 1
5	joystick 1 button 2
6	joystick 1 button 1
7	joystick 2 button 2

The button bits are 0 when the button is fired, 1 otherwise. The axis values are normally 0 but are set to 1 by `OUT` `&h201`, *x* and then fall back to 0 after a delay. The longer the delay, the higher the axis value.

port	Effect
other values	Returns zero.

Notes

- Only a limited number of machine ports are emulated in PC-BASIC.

Errors

- *port* is not in `[-32768–65535]` : `Overflow` .
- *port* has a string value: `Type mismatch` .

INPUT$

```
chars = INPUT[ ]$ (num_chars [, [#] file_num])
```

Returns a string of `num_chars` characters from the keyboard or, if `file_num` is provided, from a text file.

Parameters

- `num_chars` is a numeric expression in `[1—255]`.
- `file_num` is a numeric expression that returns the number of a text file opened in `INPUT` mode. The `#` is optional and has no effect.

Notes

- This is a blocking read. It will wait for characters if there are none in the buffer.
- All control characters except `Ctrl`+`Break`, `Ctrl`+`Scroll Lock` and `Pause` are passed to the string by `INPUT$`. `Ctrl`+`Break` and `Ctrl`+`Scroll Lock` break execution whereas `Pause` halts until another key is pressed (and not read).
- When reading from the keyboard directly or through `KYBD:`, arrow keys, `Del`, `Home`, `End`, `Pg Up`, `Pg Dn` are passed as `NUL` characters. Function keys are ignored if they are event-trapped, otherwise function-key macro replacement is active as normal.

Errors

- `num_chars` is not in `[-32768—32767]`: `Overflow`.
- `num_chars` is not in `[1—255]`: `Illegal function call`.
- `file_num` is not an open file: `Bad file number`.
- `file_num` is less than zero: `Illegal function call`.
- `file_num` is greater than `32767`: `Overflow`.
- `file_num` is not open for `INPUT`: `Bad file mode`.
- `num_chars` or `file_num` are strings: `Type mismatch`.
- `file_num` is open to a `COM` port and this is the first `INPUT`, `LINE INPUT` or `INPUT$` call on that port since the buffer has filled up completely (i.e. `LOF(file_num)` has become zero): `Communication buffer overflow`.

INSTR

```
position = INSTR([start,] parent, child)
```

Returns the location of the first occurrence of the substring `child` in `parent` .

Parameters

- `parent` and `child` are string expressions.
- `start` is a numeric expression in `[1–255]` , specifying the starting position from where to look; if not specified, the search starts at character 1.

Notes

- If `child` is not a substring of `parent` occurring at or before `start` , `INSTR` returns 0.

Errors

- `start` has a string value or `parent` or `child` have numeric values: `Type mismatch` .
- `start` is not in `[-32768–32767]` : `Overflow` .
- `start` is not in `[1–255]` : `Illegal function call` .

INT

```
whole = INT(number)
```

Returns `number` truncated towards negative infinity.

Parameters

- `number` is an expression.

Notes

- `FIX` truncates towards zero: it removes the fractional part. By contrast, `INT` truncates towards negative infinity.
- If `number` is a string expression, `INT` returns its value unchanged.

IOCTL$

```
result = IOCTL[ ]$ ([#] file_num)
```

Raises `Illegal function call` .

Notes

- In GW-BASIC, `IOCTL$` reads the reply to `IOCTL` from a device.
- This function is not implemented in PC-BASIC.

Errors

- `file_num` has a string value: `Type mismatch` .
- `file_num` is not in `[-32768–32767]` : `Overflow` .
- `file_num` is not an open file: `Bad file number` .
- Otherwise: `Illegal function call`

LEFT$

```
child = LEFT$(parent, num_chars)
```

Returns the leftmost `num_chars` characters of `parent` .

Parameters

- `parent` is a string expression.
- `num_chars` is a numeric expression in `[0–255]` .

Notes

- If `num_chars` is zero or `parent` is empty, `LEFT$` returns an empty string.
- If `num_chars` is greater than the length of `parent` , returns `parent` .

Errors

- `parent` has a numeric value or `num_chars` has a string value: `Type mismatch` .
- `num_chars` is not in `[-32768–32767]` : `Overflow` .
- `num_chars` is not in `[0–255]` : `Illegal function call` .

```
length = LEN(string)
```

Returns the number of characters in `string`.

Parameters

- `string` is a string expression.

Errors

- `string` has a number value: `Type mismatch`.

LOC

```
location = LOC(file_num)
```

Returns the current location in the file opened under number `file_num` .

- If the file is opened for `INPUT` , `OUTPUT` or `APPEND` , `LOC` returns the number of 128-byte blocks read or written since opening the file.
- If the file is opened for `RANDOM` , `LOC` returns the record number last read or written.
- If the file is opened to a `COM` device, `LOC` returns the number of characters in the input buffer, with a maximum of 255.
- If the file is opened to `KYBD:` , `LOC` returns 0.

Parameters
- `file_num` is a numeric expression in the range `[0–255]` .

Notes
- `file_num` must not be preceded by a `#` .
- In `OUTPUT` or `APPEND` mode, before any writes `LOC` returns 0. After the 128th character is written, `LOC` returns 1.
- In `INPUT` mode, before any reads `LOC` returns 1. After the **129th** character is read, `LOC` returns 2.

Errors
- `file_num` has a string value: `Type mismatch` .
- `file_num` is not in `[-32768–32767]` : `Overflow` .
- `file_num` is not in `[0–255]` : `Illegal function call` .
- `file_num` is not an open file: `Bad file number` .
- `file_num` is open to a `LPT` device: `Bad file mode` .

LOF

```
length = LOF(file_num)
```

Returns the number of bytes in the file open under `file_num`.

Parameters

- `file_num` is a numeric expression in the range `[0–255]`.

Notes

- If `file_num` is open to a `COM:` device, `LOF` returns the number of bytes free in the input buffer.

Errors

- `file_num` has a string value: `Type mismatch`.
- `file_num` is not in `[-32768–32767]`: `Overflow`.
- `file_num` is not in `[0–255]`: `Illegal function call`.
- `file_num` is not an open file: `Bad file number`.
- `file_num` is open to a `LPT` device: `Bad file mode`.

LOG

```
y = LOG(x)
```

Returns the natural logarithm of `x`.

Parameters

- `x` is a numeric expression greater than zero.

Notes

- Unless PC-BASIC is run with the `double` option, this function returns a single-precision value.
- `LOG(x)` can differ from GW-BASIC by 1 in the least significant digit.

Errors

- `x` has a string value: `Type mismatch`.
- `x` is zero or negative: `Illegal function call`.

LPOS

```
position = LPOS(printer_number)
```

Returns the column position for a printer.

Parameters

- `printer_number` is a numeric expression in `[0–3]`. If it is 0 or 1, the position for `LPT1:` is returned. If it is 2, `LPT2:`; 3, `LPT3:`.

Notes

- When entering direct mode, `LPT1:` (but not other printers) is flushed and its position is reset to 1.

Errors

- `printer_number` has a string value: `Type mismatch`.
- `printer_number` is not in `[-32768–32767]`: `Overflow`.
- `printer_number` is not in `[0–3]`: `Illegal function call`.

MID$ (function)

```
substring = MID$(string, position [, length])
```

Returns a substring of `string` starting at `position`, counting from 1. The substring has length `length` if specified. If `length` is not specified, the substring extends to the end of the string.

Parameters

- `string` is a string expression.
- `position` is a numeric expression between 1 and the string length, inclusive.
- `length` is a numeric expression in `[0–255]`.

Errors

- `string` has a number value or `position` or `length` have string values: `Type mismatch`.
- `position` or `length` are not in `[-32768–32767]`: `Overflow`.
- `position` is not in `[1–255]`: `Illegal function call`.
- `length` is not in `[0–255]`: `Illegal function call`.

MKD$

```
bytes = MKD$(double)
```

Returns the internal 8-byte Microsoft Binary Format representation of a double- precision number.

Errors

- `double` has a string value: `Type mismatch` .

MKI$

```
bytes = MKI$(int)
```

Returns the internal 2-byte little-endian representation of an integer.

Errors

- `int` has a string value: `Type mismatch` .
- `int` is not in `[-32768—32767]` : `Overflow` .

MKS$

```
bytes = MKS$(single)
```

Returns the internal 8-byte Microsoft Binary Format representation of a single- precision number.

Errors

- `single` has a string value: `Type mismatch` .

OCT$

```
octal = OCT$(x)
```

Returns a string with the octal representation of `x`.

Parameters

- `x` is a numeric expression in `[-32768–65535]`. Values for negative `x` are shown as two's-complement.

Errors

- `x` has a string value: `Type mismatch`.
- `x` is not in `[-32768–65535]`: `Overflow`.

PEEK

```
value = PEEK(address)
```

Returns the value of the memory at `segment * 16 + address` where `segment` is the current segment set with `DEF SEG`.

Parameters

- `address` is a numeric expression in `[-32768–65535]`. Negative values are interpreted as their two's complement.

Notes

- The memory is only partly emulated in PC-BASIC. See Memory model for supported addresses. Outside emulated areas, `PEEK` returns 0.
- Values for particular memory address can be preset on the command line using the `peek` option. This can be used for compatibility with old programs. These values will override video or data segment values, if they are in those locations.

Errors

- `address` has a string value: `Type mismatch`.
- `address` is not in `[-32768–65535]`: `Overflow`.

PEN (function)

```
x = PEN(mode)
```

Reads the light pen. What this function returns depends on `mode`:

mode	Return value
0	Boolean; whether the light pen has been down since last poll.
1	x coordinate of last pen down position
2	y coordinate of last pen down position
3	Boolean; whether the pen is currently down
4	x coordinate of current pen position
5	y coordinate of current pen position
6	character row coordinate of last pen down position
7	character column coordinate of last pen down position
8	character row coordinate of current pen position
9	character column coordinate of current pen position

Parameters

- `mode` is a numeric expression in `[0–9]`.

Notes

- In PC-BASIC, for *pen down* read *mouse button pressed*. For *pen position* read *mouse pointer position*.

Errors

- `mode` has a string value: `Type mismatch`.
- `mode` is not in `[-32768–32767]`: `Overflow`.
- `mode` is not in `[0–9]`: `Illegal function call`.

`PLAY` (function)

```
length = PLAY(voice)
```

Returns the number of notes in the background music queue. The return value is in `[0–32]` .

Parameters

- `voice` is a numeric expression in `[0–255]` . If `syntax={pcjr|tandy}` , indicates for which tone voice channel the number of notes is to be returned. If `voice` is not in `[0–2]` , the queue for voice 0 is returned. For other choices of `syntax` , the `voice` value has no effect.

Notes

- There are at most 32 notes in the music queue. However, unless the articulation is set to legato, there are short gaps between each note; these are counted as separate notes in the queue. Effectively, the queue length is thus 16 for the default and staccato articulations and 32 for legato.

Errors

- `voice` has a string value: `Type mismatch` .
- `voice` is not in `[0–255]` : `Illegal function call` .
- `voice` is not in `[-32768–32767]` : `Overflow` .

PMAP

```
transformed_coord = PMAP(original_coord, fn)
```

Maps between viewport and logical (`WINDOW`) coordinates. If no `VIEW` has been set, the viewport coordinates are physical coordinates.

Depending on the value of `fn` , `PMAP` transforms from logical to viewport coordinates or vice versa:

fn	Return value
0	return viewport x given logical x
1	return viewport y given logical y
2	return logical x given viewport x
3	return logical y given viewport y

Parameters
- `fn` is a numeric expression in `[0–3]` .

Notes
- Initially, in text mode, `PMAP` returns 0.
- In GW-BASIC, `PMAP` behaves anomalously on `SCREEN` changes, where it sometimes returns results as if the last `WINDOW` setting had persisted. This behaviour is not implemented in PC-BASIC.

Errors
- Any of the parameters has a string value: `Type mismatch` .
- A physical coordinate is not in `[-32768–32767]` : `Overflow` .
- `fn` is not in `[-32768–32767]` : `Overflow` .
- `fn` is not in `[0–3]` : `Illegal function call` .

`POINT` **(current coordinate)**

```
coord = POINT(fn)
```

Returns a currently active coordinate of the graphics screen. This is usually the last position at which a pixel has been plotted, the second corner given in a `LINE` command, or the centre of the viewport if nothing has been plotted. `fn` is a numeric expression in `[0–3]`.

The coordinate returned depends on the value of `fn`:

fn	**Return value**
0	viewport x
1	viewport y
2	logical x
3	logical y

Parameters

- `fn` is a numeric expression in `[0–3]`.

Notes

- In text mode, returns the active coordinate of any previous graphics mode; if no graphics mode has been active, returns 0.

Errors

- `fn` has a string value: `Type mismatch`.
- `fn` is not in `[-32768–32767]`: `Overflow`.
- `fn` is not in `[0–3]`: `Illegal function call`.

`POINT` (pixel attribute)

```
attrib = POINT(x, y)
```

Returns the attribute of the pixel at logical coordinate `x` , `y` .

Parameters

- `x` , `y` are numeric expressions in `[-32768—32767]` .

Notes

- If `x` , `y` is outside the screen, returns -1.

Errors

- Function is called in text mode: `Illegal function call` .
- `x` or `y` has a string value: `Type mismatch` .
- `x` or `y` or the physical coordinates they translate into are not in `[-32768—32767]` : `Overflow` .

`POS`

```
pos = POS(dummy)
```

Returns the current cursor column position, in the range `[1—80]` .

Parameters

- `dummy` is a valid expression of any type; its value has no effect.

RIGHT$

```
child = RIGHT$(parent, num_chars)
```

Returns the rightmost `num_chars` characters of `parent`. If `num_chars` is zero or `parent` is empty, `RIGHT$` returns an empty string. If `num_chars` is greater than the length of `parent`, returns `parent`.

Parameters

- `parent` is a string expression.
- `num_chars` is a numeric expression in `[0–255]`.

Errors

- `num_chars` has a string value: `Type mismatch`.
- `num_chars` is not in `[-32768–32767]`: `Overflow`.
- `num_chars` is not in `[0–255]`: `Illegal function call`.

---

random **= RND**[**(***x***)**]

Returns a pseudorandom number in the interval `[0–1)` .

Parameters

`x` is a numeric expression.

- If `x` is zero, `RND` repeats the last pseudo-random number.
- If `x` is greater than zero, a new pseudorandom number is returned.
- If `x` is negative, `x` is converted to a single-precision floating-point value and the random number seed is set to the absolute value of its mantissa. The function then generates a new pseudorandom numer with this seed. Since the only the mantissa of `x` is used, any two values whose ratio is a power of 2 will produce the same seed. Note that this procedure for generating a new seed differs from that used by `RANDOMIZE` .

Notes

- PC-BASIC's `RND` function generates pseudo-random numbers through a linear congruential generator with modulo 2^{24}, multiplier 214013 and increment 2531011. This exactly reproduces the random numbers of GW-BASIC's `RND` .
- It should be noted, however, that this is a very poor random number generator: its parameters imply a recurrence period of 2^{24}, meaning that after less than 17 million calls `RND` will wrap around and start running through the exact same series of numbers all over again. `RND` should not be used for cryptography, scientific simulations or anything else remotely serious.

Errors

- `x` has a string value: `Type mismatch` .

SCREEN **(function)**

```
value = SCREEN(row, column [, fn])
```

Returns the <u>code point</u> or colour attribute for the character at position `row` , `col` .

Parameters

- `row` is a numeric expression in the range `[1–25]` .
- `col` is a numeric expression between 1 and the screen width (40 or 80).
- `fn` is a numeric expression in `[0–255]` . If it is zero or not specified, the code point of the character is returned. If it is non-zero, in text mode the attribute is returned; in other screens, 0 is returned.

Errors

- Any parameter has a string value: `Type mismatch` .
- `fn` is not in `[0–255]` : `Illegal function call` .
- `fn` is not in `[-32768–32767]` : `Overflow` .
- `row` is not inside the current `VIEW PRINT` area: `Illegal function call` .
- `KEY ON` and `row=25` : `Illegal function call` .
- `col` is not in `[1, width]` : `Illegal function call` .

SGN

```
sign = SGN(number)
```

Returns the sign of `number` : `1` for positive, `0` for zero and `-1` for negative.

Parameters

- `number` is a numeric expression.

Errors

- `number` has a string value: `Type mismatch` .

SIN

```
sine = SIN(angle)
```

Returns the sine of `angle` .

Parameters

- `angle` is a numeric expression giving the angle in radians.

Notes

- Unless PC-BASIC is run with the `double` option, this function returns a single-precision value.
- The sine returned usually differs from the value returned by GW-BASIC in the least significant figure.

Errors

- `angle` has a string value: `Type mismatch` .

SPACE$

```
spaces = SPACE$(number)
```

Returns a string of `number` spaces.

Parameters

- `number` is a numeric expression in `[0–255]` .

Errors

- `number` has a string value: `Type mismatch` .
- `number` is not in `[-32768–32767]` : `Overflow` .
- `number` is not in `[0–255]` : `Illegal function call` .

SQR

```
root = SQR(number)
```

Returns the square root of `number` .

Parameters

- `number` is a numeric expression.

Notes

- Unless PC-BASIC is run with the `double` option, this function returns a single-precision value.

Errors

- `number` has a string value: `Type mismatch`

STICK

```
pos = STICK(axis)
```

Returns a coordinate of a joystick axis. All coordinates returned are in the range `[1—254]` with `128` indicating the neutral position.

axis	Return value
0	1st joystick x coordinate
1	1st joystick y coordinate
2	2nd joystick x coordinate
3	2nd joystick y coordinate

Parameters

- `axis` is a numeric expression in `[0—3]` and indicates which axis to read.

Errors

- `axis` has a string value: `Type mismatch`
- `axis` is not in `[-32768—32767]` : `Overflow` .
- `axis` is not in `[0—3]` : `Illegal function call` .

STR$

```
repr = STR$(number)
```

Returns the string representation of `number` .

Parameters

- `number` is a numeric expression.

Errors

- `number` has a string value: `Type mismatch` .

STRIG (function)

```
result = STRIG(mode)
```

Returns the status of the joystick trigger buttons. `STRIG` returns the following results, all Boolean values:

mode	Return value
0	1st joystick, 1st trigger has been pressed since last poll.
1	1st joystick, 1st trigger is currently pressed.
2	2nd joystick, 1st trigger has been pressed since last poll.
3	2nd joystick, 1st trigger is currently pressed.
4	1st joystick, 2nd trigger has been pressed since last poll.
5	1st joystick, 2nd trigger is currently pressed.
6	2nd joystick, 2nd trigger has been pressed since last poll.
7	2nd joystick, 2nd trigger is currently pressed.

Parameters

- `mode` is a numeric expression in `[0–7]` .

Notes

- The `STRIG` function returns correct results regardless of the `STRIG ON` status or whether `STRIG(0)` has been called first.

Errors

- `mode` has a string value: `Type mismatch` .
- `mode` is not in `[-32768–32767]` : `Overflow` .
- `mode` is not in `[0–7]` : `Illegal function call` .

STRING$

```
string = STRING$(length, char)
```

Returns a string of `length` times character `char` .

Parameters

- If `char` is a numeric expression, it must be in `[0–255]` and is interpreted as the <u>code point</u> of the character.
- If `char` is a string expression, its first character is used.

Errors

- `length` has a string value: `Type mismatch` .
- `char` is the empty string: `Illegal function call` .
- `char` or `length` is not in `[-32768–32767]` : `Overflow` .
- `char` or `length` is not in `[0–255]` : `Illegal function call` .

TAN

```
tangent = TAN(angle)
```

Returns the tangent of `angle` .

Parameters

- `angle` is a numeric expression giving the angle in radians.

Notes

- Unless PC-BASIC is run with the `double` option, this function returns a single-precision value.
- The tangent returned usually differs from the value returned by GW-BASIC in the least significant figure.
- For `angle` close to multiples of `π/2` , the tangent is divergent or close to zero. The values returned will have very low precision in these cases.

Errors

- `angle` has a string value: `Type mismatch` .

TIME$ (function)

```
time = TIME$
```

Returns the current BASIC time in the form `"HH:mm:ss"`.

Notes

- This function takes no arguments.

TIMER (function)

```
seconds = TIMER
```

Returns the number of seconds since midnight on the internal BASIC clock.

Notes

- `TIMER` updates in ticks of 1/20 second.
- The least-significant two bytes of `TIMER` are often used as a seed for the pseudorandom number generator through `RANDOMIZE TIMER`. Since these bytes only take values from a limited set, that's not in fact a particularly good random seed. However, the pseudorandom number generator included with GW-BASIC and PC-BASIC is so weak that it should not be used for anything serious anyway.
- This function takes no arguments.

USR

```
value = USR[n](expr)
```

Raises `Illegal function call`.

Parameters

- `n` is a digit `[0−9]`.
- `expr` is an expression.

Notes

- In GW-BASIC, calls a machine-code function and returns its return value.
- This function is not implemented in PC-BASIC.

Errors

- `n` is not a digit `[0−9]`: `Syntax error`.

VAL

`value = VAL(string)`

Returns the numeric value of the string expression `string` . Parsing stops as soon as the first character is encountered that cannot be part of a number. If no characters are parsed, `VAL` returns zero. See the section on <u>numeric literals</u> for the recognised number formats.

Notes

- Spaces before or even inside a number are ignored: `VAL(" 1 0")` returns `10` .
- If `string` contains one of the ASCII separator characters `CHR$(28)` (file separator), `CHR$(29)` (group separator) or `CHR$(31)` (unit separator), `VAL` returns zero. This is not the case with `CHR$(30)` (record separator). This behaviour conforms to GW-BASIC.

Errors

- `string` has a number value: `Type mismatch` .

VARPTR

`pointer = VARPTR({name|#file_num})`

Returns the memory address of variable `name` or of the File Control Block of file number `file_num` .

Parameters

- `name` is a previously defined variable or fully indexed array element.
- `file_num` is a legal file number.

Notes

- `VARPTR` can be used with `PEEK` to read a variable's internal representation.

Errors

- `name` has not been previously defined: `Illegal function call` .
- `file_num` has a string value: `Type mismatch` .
- `file_num` is not in `[1, max_files]` , where `max_files` is the maximum number of files as set by the `max-files` option: `Bad file number` .

VARPTR$

```
pointer = VARPTR$(name)
```

Returns the memory address of variable `name` in the form of a 3-byte string. The first byte is the length of the record the pointer points to:

2	for integers
3	for strings (length + pointer to string space)
4	for single-precision floats
8	for double-precision floats

The last two bytes are the pointer address (as returned by `VARPTR`) in little-endian order.

Errors

- `name` has not been previously defined: `Illegal function call` .

6.7. Statements

A program line is composed of a line number and one or more *statements*. If multiple statements are put on one line, they must be separated by colons `:` . Statements may be empty. Each statement has its own idiosyncratic syntax.

Many reference works on GW-BASIC distinguish *commands* and statements; this distinction stems from the original Dartmouth design of the BASIC language, in which commands were not part of the language and could not be used in programs, but were rather used to control the interpreter itself. However, in GW-BASIC this distinction is less useful and therefore this reference includes what is traditionally thought of as commands in the category of statements.

AUTO

```
AUTO [line_number|.] [, [increment]]
```

Start automatic line numbering. Line numbers are automatically generated when `Enter` is pressed. If a program line exists at a generated line number, a `*` is shown after the line number. To avoid overwriting this line, leave it empty and press `Enter`. To stop automatic line numbering, press `Ctrl`+`Break` or `Ctrl`+`C`. The line being edited at that point is not saved. BASIC will return to command mode, even if `AUTO` was run from a program line.

Parameters
- Line numbering starts at `line_number`, if specified. If `.` is specified, line numbering starts at the last program line that was stored. Otherwise, line numbering starts at `10`.
- Each next line number is incremented by `increment`, if specified. If a comma is used without specifying an increment, the last increment specified in an `AUTO` command is used. If not, `increment` defaults to `10`.

Errors
- `line_number` is not an unsigned-integer value in `[0–65529]` : `Syntax error`.
- When automatic line numbering is enabled and `Enter` is pressed on an empty line with number larger than `65519` : `Undefined line number`.
- `increment` is `0` : `Illegal function call`.

BEEP

`BEEP`

Beep the speaker at 800Hz for 0.25s.

Errors

- If a `Syntax error` is raised, the beep is still produced.

`BEEP` (switch)

`BEEP {ON|OFF}`

Switches the internal speaker on or off.

Notes

- Only legal with the `syntax={pcjr|tandy}` option.
- On PC-BASIC, both the internal and the external speaker are emulated through the same sound system.

BLOAD

`BLOAD file_spec [, offset]`

Loads a memory image file into memory.

Parameters

- The string expression *file_spec* is a valid file specification indicating the file to read the memory image from.
- *offset* is a numeric expression in the range `[-32768–65535]`. It indicates an offset in the current `DEF SEG` segment where the file is to be stored. If not specified, the offset stored in the `BSAVE` file will be used. If negative, its two's complement will be used.

Errors

- The loaded file is not in `BSAVE` format: `Bad file mode`.
- *file_spec* contains disallowed characters: `Bad file number` (on `CAS1:`); `Bad file name` (on disk devices).
- *file_spec* has a numeric value: `Type mismatch`.
- *offset* is not in the range `[-32768–65535]`: `Overflow`.

BSAVE

```
BSAVE file_spec, offset, length
```

Saves a region of memory to an image file.

Parameters

- The string expression `file_spec` is a valid <u>file specification</u> indicating the file to write to.
- `offset` is a numeric expression in the range `[-32768–65535]` indicating the offset into the current `DEF SEG` segment from where to start reading.
- `length` is a numeric expression in the range `[-32768–65535]` indicating the number of bytes to read.
- If `offset` or `length` are negative, their two's complement will be used.

Errors

- `file_spec` has a numeric value: `Type mismatch`.
- `file_spec` contains disallowed characters: `Bad file number` (on `CAS1:`); `Bad file name` (on disk devices).
- `offset` is not in the range `[-32768–65535]`: `Overflow`.
- `length` is not in the range `[-32768–65535]`: `Overflow`.

CALL and CALLS

```
{CALL|CALLS} address_var [( p0 [, p1] ... )]
```

Does nothing.

Notes

- In GW-BASIC, `CALL` or `CALLS` executes a machine language subroutine.
- This statement is not implemented in PC-BASIC.

Parameters

- `address_var` is a numeric variable name.
- `p0`, `p1`, `...` are variable names or array elements.

Errors

- `address_var` is a string variable: `Type mismatch`.
- `address_var` is a literal or expression: `Syntax error`.

CHAIN

```
CHAIN [MERGE] file_spec [, [line_number_expr] [, ALL] [, DELETE range [, ign]]]
```

Loads a program from file into memory and runs it, optionally transferring variables.

- If `ALL` is specified, all variables are transferred. If not, the variables specified in a `COMMON` statement are transferred.
- If `MERGE` is specified, the loaded program is merged into the existing program. To be able to use this, the program file indicated by `file_spec` must be in plain text format.
- If `DELETE` is specified, the `range` of line numbers is deleted from the existing code before the merge. This is pointless without `MERGE`.

Parameters

- The string expression `file_spec` is a valid <u>file specification</u> indicating the file to read the program from.
- `line_number_expr` is a numeric expression. It will be interpreted as a line number in the new program and execution will start from this line number. If `line_number_expr` is negative, it will be interpreted as its two's-complement.
- `range` is a line number range of which the closing line number is specified and exists before the merge.
- `ign` is optional and ignored.

Notes

- `CHAIN` preserves the `OPTION BASE` setting.
- Only if `ALL` is specified, `DEF FN` definitions are preserved.
- Only if `MERGE` is specified, `DEFINT`, `DEFSTR`, `DEFSNG`, `DEFDBL` definitions are preserved.
- If specified, `ALL` must precede `DELETE`; if unspecified, no comma must be put in its place and only two commas should precede `DELETE`.

Errors

- `file_spec` has a numeric value: `Type mismatch`.
- `file_spec` contains disallowed characters: `Bad file number` (on `CAS1:`); `Bad file name` (on disk devices).
- The file specified in `file_spec` cannot be found: `File not found`.
- `MERGE` is specified and the loaded program was not saved in plain-text mode: `Bad file mode`.
- A line number in `range` is greater than 65529: `Syntax error`.

- If a `Syntax error` is raised by `CHAIN`, no lines are deleted and the new program is not loaded.
- The closing line number in `range` does not exist: `Illegal function call`
- If `line_number_expr` does not evaluate to an existing line number in the new program, `Illegal function call` is raised but the load or merge is being performed.
- A loaded text file contains lines without line numbers: `Direct statement in file`.
- A loaded text file contains lines longer than 255 characters: `Line buffer overflow`. Attempting to load a text file that has `LF` rather than `CR LF` line endings may cause this error.

CHDIR

```
CHDIR dir_spec
```

Change the current directory on a disk device to `dir_spec`. Each disk device has its own current directory.

Parameters

- The string expression `dir_spec` is a valid file specification indicating an existing directory on a disk device.

Errors

- No matching path is found: `Path not found`.
- `dir_spec` has a numeric value: `Type mismatch`.
- `dir_spec` is empty: `Bad file name`.

CIRCLE

```
CIRCLE [STEP] (x, y), radius [, [colour] [, [start] [, [end] [, aspect]]]
```

Draw an ellipse or ellipse sector.

Parameters

- The midpoint of the ellipse is at `(x, y)`. If `STEP` is specified, the midpoint is `(x, y)` away from the current position.
- `radius` is the radius, in pixels, along the long axis.
- `colour` is the colour attribute.
- If `start` and `end` are specified, a sector of the ellipse is drawn from `start` radians to `end` radians, with zero radians the intersection with the right-hand x axis. If a negative value is specified, the arc sector is connected by a line to the midpoint.
- `aspect` specifies the ratio between the y radius and the x radius. If it is not specified, the standard value for the `SCREEN` mode is used (see there), so as to make the ellipse appear like a circle on the original hardware.

Notes

- For `aspect <> 1`, the midpoint algorithm used does not pixel-perfectly reproduce GW-BASIC's ellipses.

Errors

- The statement is executed in text mode: `Illegal function call`.
- `start` or `end` is not in `[0−2π]`: `Illegal function call`.
- The statement ends with a comma: `Missing operand`.

CLEAR

```
CLEAR [expr] [, [mem_limit] [, [stack_size] [, video_memory]]]
```

Clears all variables, arrays, `DEF FN` user functions and `DEF`*type* type definitions. Closes all files. Turns off all sound. Resets `PLAY` state and sets music to foreground. Clears all `ON ERROR` traps. Resets `ERR` and `ERL` to zero. Disables all events. Turns `PEN` and `STRIG` off. Resets the random number generator. Clears the loop stack. Resets the `DRAW` state and the current graphics position.

Parameters

- `mem_limit` specifies the upper limit of usable memory. Default is previous memory size. Default memory size is 65534.
- `stack_size` specifies the amount of memory available to the BASIC stack. Default is previous stack size. Default stack size is 512.
- `video_memory` specifies the amount of memory available to the video adapter. This parameter is only legal with one of the options `syntax={pcjr, tandy}`. Instead of using `CLEAR`, the option `video-memory` can also be used to set video memory size.

Notes

- The purpose of `expr` is unknown.
- If called inside a `FOR` — `NEXT` or `WHILE` — `WEND` loop, an error will be raised at the `NEXT` or `WEND` statement, since the loop stacks have been cleared.

Errors

- Any of the arguments has a string value: `Type mismatch`.
- `mem_limit`, `stack_size` are not in `[-32768–65535]`: `Overflow`.
- `mem_limit` or `stack_size` equal `0`: `Illegal function call`.
- `mem_limit` equals `-1` or `65535`: `Out of memory`.
- `mem_limit` or `expr` are too low: `Out of memory`.
- `expr` is not in `[-32768–32767]`: `Overflow`.
- `expr` is negative: `Illegal function call`.

CLOSE

```
CLOSE [[#] file_0 [, [#] file_1] ...]
```

Closes files. If no file numbers are specified, all open files are closed. The hash (`#`) is optional and has no effect.

Parameters

- `file_1, file_2, ...` are numeric expressions yielding file numbers.

Notes

- No error is raised if the specified file numbers were not open.

Errors

- `file_1, file_2, ...` are not in `[-32768—32767]` : `Overflow` .
- `file_1, file_2, ...` are not in `[0—255]` : `Illegal function call` .
- `file_1, file_2, ...` have a string value: `Type mismatch` .
- The statement ends in a comma, `Missing operand` .
- If an error occurs, only the files before the erratic value are closed.

CLS

```
CLS [x][,]
```

Clears the screen or part of it. If `x` is not specified, in SCREEN 0 the text view region is cleared; in other screens, the graphics view region is cleared. The comma is optional and has no effect.

Parameters

`x` is a numeric valued expression.

- If `x = 0`, the whole screen is cleared.
- If `x = 1`, the graphics view region is cleared.
- If `x = 2`, the text view region is cleared.

The optional argument `x` is not available with `syntax={pcjr|tandy}`.

Errors

- `x` is has a string value: `Type mismatch`.
- `x` is not in `[-32768—32767]` : `Overflow` .
- `x` is not in `[0, 1, 2]` : `Illegal function call` .
- No comma is specified but more text follows: `Illegal function call` .
- A comma is specified followed by more: `Syntax error` .
- `syntax=pcjr` is set and an argument is specified: `Syntax error` .
- `syntax=tandy` is set and an argument is specified: `Illegal function call` .
- If an error occurs, the screen is not cleared.

`COLOR` **(text mode)**

```
COLOR [foreground] [, [background] [, border]]
```

Changes the current foreground and background attributes. All new characters printed will take the newly set attributes. Existing characters on the screen are not affected.

Parameters

- `foreground` is a numeric expression in `[0–31]`. This specifies the new foreground attribute. Attributes `16–31` are blinking versions of attributes `0–15`.
- `background` is a numeric expression in `[0–15]`. This specifies the new background attribute. It is taken `MOD 8`: Values `8–15` produce the same colour as `0–7`.
- `border` is a numeric expression in `[0–15]` specifying the border attribute.

Textmode attributes (colour)

FG	Background attribute							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	00	10	20	30	40	50	60	70
1	01	11	21	31	41	51	61	71
2	02	12	22	32	42	52	62	72
3	03	13	23	33	43	53	63	73
4	04	14	24	34	44	54	64	74
5	05	15	25	35	45	55	65	75
6	06	16	26	36	46	56	66	76
7	07	17	27	37	47	57	67	77
8	08	18	28	38	48	58	68	78
9	09	19	29	39	49	59	69	79
10	0a	1a	2a	3a	4a	5a	6a	7a
11	0b	1b	2b	3b	4b	5b	6b	7b
12	0c	1c	2c	3c	4c	5c	6c	7c
13	0d	1d	2d	3d	4d	5d	6d	7d
14	0e	1e	2e	3e	4e	5e	6e	7e
15	0f	1f	2f	3f	4f	5f	6f	7f
16	80	90	a0	b0	c0	d0	e0	f0
17	81	91	a1	b1	c1	d1	e1	f1
18	82	92	a2	b2	c2	d2	e2	f2
19	83	93	a3	b3	c3	d3	e3	f3
20	84	94	a4	b4	c4	d4	e4	f4
21	85	95	a5	b5	c5	d5	e5	f5
22	86	96	a6	b6	c6	d6	e6	f6
23	87	97	a7	b7	c7	d7	e7	f7
24	88	98	a8	b8	c8	d8	e8	f8
25	89	99	a9	b9	c9	d9	e9	f9
26	8a	9a	aa	ba	ca	da	ea	fa
27	8b	9b	ab	bb	cb	db	eb	fb
28	8c	9c	ac	bc	cc	dc	ec	fc
29	8d	9d	ad	bd	cd	dd	ed	fd
30	8e	9e	ae	be	ce	de	ee	fe
31	8f	9f	af	bf	cf	df	ef	ff

Textmode attributes (monochrome)

		Background attribute							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
FG	**0**	00	10	20	30	40	50	60	70
	1	01	11	21	31	41	51	61	71
	2	02	12	22	32	42	52	62	72
	3	03	13	23	33	43	53	63	73
	4	04	14	24	34	44	54	64	74
	5	05	15	25	35	45	55	65	75
	6	06	16	26	36	46	56	66	76
	7	07	17	27	37	47	57	67	77
	8	08	18	28	38	48	58	68	78
	9	09	19	29	39	49	59	69	79
	10	0a	1a	2a	3a	4a	5a	6a	7a
	11	0b	1b	2b	3b	4b	5b	6b	7b
	12	0c	1c	2c	3c	4c	5c	6c	7c
	13	0d	1d	2d	3d	4d	5d	6d	7d
	14	0e	1e	2e	3e	4e	5e	6e	7e
	15	0f	1f	2f	3f	4f	5f	6f	7f
	16	80	90	a0	b0	c0	d0	e0	f0
	17	81	91	a1	b1	c1	d1	e1	f1
	18	82	92	a2	b2	c2	d2	e2	f2
	19	83	93	a3	b3	c3	d3	e3	f3
	20	84	94	a4	b4	c4	d4	e4	f4
	21	85	95	a5	b5	c5	d5	e5	f5
	22	86	96	a6	b6	c6	d6	e6	f6
	23	87	97	a7	b7	c7	d7	e7	f7
	24	88	98	a8	b8	c8	d8	e8	f8
	25	89	99	a9	b9	c9	d9	e9	f9
	26	8a	9a	aa	ba	ca	da	ea	fa
	27	8b	9b	ab	bb	cb	db	eb	fb
	28	8c	9c	ac	bc	cc	dc	ec	fc
	29	8d	9d	ad	bd	cd	dd	ed	fd
	30	8e	9e	ae	be	ce	de	ee	fe
	31	8f	9f	af	bf	cf	df	ef	ff

Notes

- The syntax and effect of `COLOR` is different in different `SCREEN` modes: <u>COLOR</u> (text mode), <u>COLOR (SCREEN 1)</u>, (<u>SCREEN 3—9</u>).
- At least one parameter must be provided and the statement must not end in a comma.

Errors

- Any of the parameters has a string value: `Type mismatch`.
- Any of the parameters is not in `[-32768—32767]`: `Overflow`.
- *foreground* is not in `[0—31]`, *background* is not in `[0—15]` or *border* is not in `[0—15]`: `Illegal function call`.
- Statement is used in `SCREEN 2`: `Illegal function call`.

COLOR (SCREEN 1)

`COLOR [palette_0] [, palette [, override]]`

Assigns new colours to the palette of attributes.

- `palette_0` is a numeric expression in `[0–255]`. This sets the palette colour associated with attribute 0; by default, the background has this attribute. All pixels with this attribute will change colour. The palette colour value is taken from the 64-colour set. `palette_0` is taken `MOD 64`.
- `palette` is a numeric expression in `[0–255]` that specifies the palette:
  - `palette` odd sets the standard CGA palette (cyan, magenta, grey).
  - `palette` even sets the alternative palette (green, red, brown).
  All pixels with attributes 1,2,3 will change colour to the new palette.
- `override` is a numeric expression in `[0–255]`. If `override` is specified, palette is set as above but using `override` instead of `palette`. `palette` is then ignored.

CGA palettes

Attribute	Palette 0			Palette 1			Alternate palette		
	Colour	Lo	Hi	Colour	Lo	Hi	Colour	Lo	Hi
0	Black			Black			Black		
1	Green			Cyan			Cyan		
2	Red			Magenta			Red		
3	Brown			White			White		

Notes

- The syntax and effect of `COLOR` is different in different `SCREEN` modes: COLOR (text mode), COLOR (SCREEN 1), (SCREEN 3–9).
- At least one parameter must be provided and the statement must not end in a comma.

Errors

- Any of the parameters has a string value: `Type mismatch`.
- Any of the parameters is not in `[-32768–32767]`: `Overflow`.
- Any of the parameters is not in `[0–255]`: `Illegal function call`.

COLOR (SCREEN 3–9)

`COLOR [foreground] [, palette_0 [, dummy]]`

Changes the current foreground attribute and the colour for attribute 0.

Parameters

- `foreground` is a numeric expression in `[1–15]` This sets the new foreground attribute. This applies only to new characters printed or pixels plotted.
- `palette_0` is a numeric expression in `[0–15]` This sets the colour associated with attribute 0; by default, the background has this attribute. All pixels with this attribute will change colour. In `SCREEN 7` and `8`, the `palette_0` colour is taken from the first 8 of the 16-colour EGA set. `palette_0` is taken `MOD 8`. IN `SCREEN 9`, the colour value is taken from the 64-colour set.
- `dummy` is a numeric expression with a value in `[0–255]` The value of `dummy` is ignored.

EGA default palette

Attribute	Colour	
0	Black	
1	Blue	
2	Green	
3	Cyan	
4	Red	
5	Magenta	
6	Brown	
7	Low-intensity white	
8	Grey	
9	Light Blue	
10	Light Green	
11	Light Cyan	
12	Light Red	
13	Light Magenta	
14	Light Yellow	
15	High-intensity white	

EGA colour list

0		8		16		24		32		40		48		56	
1		9		17		25		33		41		49		57	
2		10		18		26		34		42		50		58	
3		11		19		27		35		43		51		59	
4		12		20		28		36		44		52		60	
5		13		21		29		37		45		53		61	
6		14		22		30		38		46		54		62	
7		15		23		31		39		47		55		63	

Notes

- The syntax and effect of `COLOR` is different in different `SCREEN` modes: COLOR (text mode), COLOR (`SCREEN 1`), (`SCREEN 3–9`).
- At least one parameter must be provided and the statement must not end in a comma.

Errors

- Any of the parameters has a string value: `Type mismatch`.
- Any of the parameters is not in `[-32768–32767]`: `Overflow`.
- *foreground* is not in `[1–15]`; *background* is not in `[0–15]`; or *dummy* is not in `[0–255]`: `Illegal function call`.

COM

`COM(`*`port`*`)` `{`**`ON`**`|`**`OFF`**`|`**`STOP`**`}`

- `ON` : enables `ON COM(`*`port`*`)` event trapping of the emulated serial port.
- `OFF` : disables trapping.
- `STOP` : halts trapping until `COM(`*`port`*`) ON` is used. Events that occur while trapping is halted will trigger immediately when trapping is re-enabled.

Parameters

- *`port`* is a numeric expression with a value of `1` or `2` . This specifies which serial port (`COM1:` or `COM2:`) is trapped. If *`port`* equals `0` , this statement does nothing.

Errors

- *`port`* a string value: `Type mismatch` .
- *`port`* is not in `[-32768—32767]` : `Overflow` .
- *`port`* is not in `[0—3]` : `Illegal function call` .

COMMON

```
COMMON [var_0 [( [index_0] )] [, [var_1 [( [index_1] )] ]] ...]
```

Specifies variables to be passed as common variables to the program called with `CHAIN` .

Parameters

- `var_0`, `var_1`, `...` are names of scalar or array variables.
- `index_0`, `index_1`, `...` are optional number literals; they are ignored.

Notes

- Array elements with square brackets and an index do not cause an error, but are ignored.
- `COMMON` statements are not executed during run time; rather, when a `CHAIN` command is encountered where `ALL` is not specified, all `COMMON` declarations in the program are parsed. As a consequence, the `DEFSTR` , `DEFINT` , `DEFSNG` or `DEFDBL` settings used are those that are active at the time of execution of the `CHAIN` statement.
- `COMMON` declarations need not be reachable in the program flow in order to be used. They may occur anywhere before or after the `CHAIN` statement that uses them.
- Variables may be repeated or occur in multiple `COMMON` declarations.
- If the `COMMON` keyword is not the first element of the statement, the declaration will be ignored. In particular, any `COMMON` declaration that occurs directly after a `THEN` or `ELSE` keyword will not be used. `COMMON` in the second or later statements of a compound statement after `THEN` or `ELSE` will be used regardless of the value of the `IF` condition.

CONT

```
CONT [anything]
```

Resumes execution of a program that has been halted by `STOP` , `END` , `Ctrl`+`C`, or `Ctrl`+`Break`.

Notes

- Anything after the `CONT` keyword is ignored.
- This statement can only be used in direct mode.
- If a break is encountered in `GOSUB` routine called from a continuing direct line (e.g. `GOSUB 100:PRINT A$`), `CONT` will overwrite the running direct line. As the subroutine `RETURN`s to the position after the `GOSUB` in the old direct line, strange things may happen if commands are given after `CONT` . In GW-BASIC, this can lead to strange errors in non-existing program lines as the parser executes bytes that are not part of a program line. In PC-BASIC, if the new direct line is shorter, execution stops after `RETURN` ; but if the direct line is extended beyond the old return position, the parser tries to resume at that return position, with strange effects.

Errors

- No program is loaded, a program has not been run, after a program line has been modified or after `CLEAR` : `Can't continue` .
- The break occurred in a direct line: `Can't continue` .
- `CONT` is used in a program: `Can't continue` .

DATA

```
DATA [const_0] [, [const_1]] ...
```

Specifies data that can be read by a `READ` statement.

Parameters

- `const_0, const_1, ...` are string and number literals or may be empty. String literals can be given with or without quotation marks. If quotation marks are omitted, leading and trailing whitespace is ignored and commas or colons will terminate the data statement.

Notes

- `DATA` declarations need not be reachable in the program flow in order to be used. They may occur anywhere before or after the `READ` statement that uses them.
- If the `DATA` keyword is not the first element of the statement, the declaration will be ignored. In particular, any `DATA` declaration that occurs directly after a `THEN` or `ELSE` keyword will not be used. `DATA` in the second or later statements of a compound statement after `THEN` or `ELSE` will be used regardless of the value of the `IF` condition.

Errors

- If the type of the literal does not match that of the corresponding `READ` statement, a `Syntax error` occurs on the `DATA` statement.

`DATE$` **(statement)**

`DATE$ = `*date*

Sets the system date. `date` is a string expression that represents a date in one of the formats: `"mm{-|/}dd{-|/}yy"` or `"mm{-|/}dd{-|/}yyyy"`

Of these,

- `mm` may be one or two characters long and must be in `[1–12]` .
- `dd` may be one or two characters long and must be in `[1–31]` .
- `yyyy` must be in `[1980–2099]` .
- `yy` may be one or two characters long and must be in one of the ranges:
    - `[0–77]` , interpreted as `2000–2077` ; or
    - `[80–99]` , interpreted as `1980–1999` .

Notes

- The system date is not actually changed; rather, PC-BASIC remembers the offset from the true system date. This avoids requiring user permission to change the system time.
- GW-BASIC appears to accept invalid dates such as `"02-31-2000"` . PC-BASIC raises `Illegal function call` for these.

Errors

- `date` has a numeric value: `Type mismatch` .
- `date` is not in the format specified above: `Illegal function call` .

DEF FN

```
DEF FN[ ]name [(arg_0 [, arg_1] ...)] = expression
```

Defines a function called `FNname` (or `FN name` : spaces between `FN` and `name` are optional). On calling `FNname( ... )` , `expression` is evaluated with the supplied parameters substituted. Any variable names used in the function that are not in the argument list refer to the corresponding global variables. The result of the evaluation is the return value of `FNname` . The type of the return value must be compatible with the type indicated by `name` .

Notes

- This statement may only be used on a program line.
- As the function must be a single expression and PC-BASIC does not have a ternary operator, there is no way to define a recursive function that actually terminates.

Parameters

- `name` must be a legal variable name.
- `arg_0`, `arg_1`, `...` must be legal variable names. These are the parameters of the function. Variables of the same name may or may not exist in the program; their value is not affected or used by the defined function.
- `expression` must be a legal PC-BASIC expression.

Errors

- The statement is executed directly instead of in a program line: `Illegal direct` .
- If the type of the return value is incompatible with the type of `name` , no error is raised at the `DEF FN` statement; however, a `Type mismatch` will be raised at the first call of `FNname` .

`DEFINT, DEFDBL, DEFSNG, DEFSTR`

`{DEFINT|DEFDBL|DEFSNG|DEFSTR}` *first_0*`[- `*last_0*`] [, `*first_1*`[- `*last_1*`]] ...`

Sets the type that is assumed if no <u>sigil</u> is specified when a variable name is used. The statement sets the default type for variables starting with a letter from the ranges specified.

The default type is set to:

`DEFINT`	integer (`%`)
`DEFDBL`	double (`#`)
`DEFSNG`	single (`!`)
`DEFSTR`	string (`$`)

Parameters

- *first_0*, *last_0*, `...` are letters of the alphabet. Pairs of letters connected by a dash `-` indicate inclusive ranges.

Notes

- `DEFSNG A-Z` is the default setting.

`DEF SEG`

`DEF SEG` `[= `*address*`]`

Sets the memory segment to be used by `BLOAD` , `BSAVE` , `CALL` , `PEEK` , `POKE` , and `USR` .

Parameters

- *address* is a numeric expression in `[-32768—65535]` .

Notes

- If *address* is negative, it is interpreted as its two's complement.
- If *address* is not specified, the segment is set to the GW-BASIC data segment.

Errors

- *address* has a string value: `Type mismatch` .
- *address* is not in `[-32768—65535]` : `Overflow` .

DEF USR

```
DEF USR[n] = address
```

Does nothing.

Parameters

- *n* is a digit between 0 and 9 inclusive.
- *address* is a numeric expression in `[-32768−65535]`.

Notes

- In GW-BASIC, this statement sets the starting address of an assembly-language function.
- This statement is not implemented in PC-BASIC.
- If *address* is negative, it is interpreted as its two's complement.

Errors

- *n* is not a digit in `[0−9]`: `Syntax error`.
- *address* has a string value: `Type mismatch`.
- *address* is not in `[-32768−65535]`: `Overflow`.

DELETE

```
DELETE [line_number_0|.] [-[line_number_1|.] ]
```

Deletes a range of lines from the program. Also stops program execution and returns control to the user.

Parameters

- *line_number_0* and *line_number_1* are line numbers in the range `[0−65529]`, specifying the inclusive range of line numbers to delete.
- A `.` indicates the last line edited.
- If the start point is omitted, the range will start at the start of the program.
- If the end point is omitted, the range will end at the end of the program.
- If no range is specified, the whole program will be deleted.

Errors

- *line_number_0* or *line_number_1* is greater than `65529`: `Syntax error`.
- The range specified does not include any program lines stored: `Illegal function call`.

DIM

```
DIM name [{(|[} limit_0 [, limit_1] ... {)|]}] [, ... ]
```

Allocates memory for one or more arrays. The `DIM` statement also fixes the number of indices of the array. An array can only be allocated once; to re-allocate an array, `ERASE` or `CLEAR` must be executed first. If an array is first used without a `DIM` statement, it is automatically allocated with its maximum indices set at `10` for each index position used. A `DIM` entry with no brackets and no indices performs no operation. Empty brackets are not allowed. The least index allowed is determined by `OPTION BASE`.

Parameters

- `name, ...` are legal variable names specifying the arrays to be allocated.
- `limit_0`, `limit_1`, `...` are numeric expressions that specify the greatest index allowed at that position.

Notes

- Mixed brackets are allowed.
- The size of arrays is limited by the available BASIC memory.
- The maximum number of indices is, theoretically, `255`. In practice, it is limited by the 255-byte limit on the length of program lines.

Errors

- `name` has already been dimensioned: `Duplicate definition`.
- An index is empty: `Syntax error`.
- An index is missing at the end: `Missing operand`.
- `limit_0`, `limit_1`, `...` have a string value: `Type mismatch`.
- `limit_0`, `limit_1`, `...` are not within `[-32768–32767]`: `Overflow`.
- `limit_0`, `limit_1`, `...` are negative: `Illegal function call`.
- The array exceeds the size of available variable space: `Out of memory`.

```
DRAW gml_string
```

Draws the shape specified by `gml_string`, a string expression in Graphics Macro Language (GML).

Graphics Macro Language reference

Movement commands

```
[B][N] movement
```

where the default is to move and draw; the optional prefixes mean:

B	move but do not plot
N	return to original point after move

and `movement` is one of:

U[n]	up n steps
L[n]	left n steps
D[n]	down n steps
R[n]	right n steps
E[n]	up and right n steps
F[n]	down and right n steps
G[n]	down and left n steps
H[n]	up and left n steps
M{+|-}x,[+|-]y	move (x,y) steps
Mx,y	move to view region coordinate (x,y)

where `n` is an integer in `[-32768–32767]` and `x`, `y` are integers in `[0–9999]`. Where optional, `n` defaults to `1`.

Scale commands

Sn	set the step size to $n/4$. The default step size is 1 pixel. n is an integer in `[1–255]`
TAn	set the angle to n degrees. The default angle is 0 degrees. n is an integer in `[-360–360]`
An	set the angle to 0 for $n=0$, 90 for $n=1$, 180 for $n=2$, 270 for $n=3$. n is an integer in `[0–3]`

Colour commands

`C`*n*	set the foreground attribute to *n*, where *n* is an integer in `[-32768–32767]` See `COLOR`.
`P`*n*,*b*	flood fill with attribute *n* and boundary attribute *b*, where *n*, *b* are integers in `[0–9999]` See `PAINT`.

Subroutine command

`X`*s*	execute a substring

`s` is one of the following:

- a string variable name followed by semicolon (`;`)
- the result of `VARPTR$()` on a string variable

Numeric variables `n` , `x` , `y` , `b` in the commands above can be:

- an integer literal, e.g. `DRAW "U100"`
- a numeric variable name or array element `var` preceded by `=` and followed by `;` . For example, `DRAW "U=VAR;"` or `DRAW "U=A(1);"`
- the result of `VARPTR$(var)` preceded by `=` . For example, `DRAW "U=" + VARPTR$(VAR)`

Notes

- The `CLS` statement resets the step size to 1 pixel, angle to 0 degrees and position to the centre of the view region.
- The value `n` in the `TA` , `A` and `C` command can be left out but *only* if the command is terminated by a semicolon. `n` defaults to `0` .
- In GW-BASIC, the numeric arguments of `U` , `L` , `D` , `R` , `E` , `F` , `G` , `H` , and `C` can be in the range `[-99999–99999]` ; however, results for large numbers are unpredictable. This is not implemented in PC-BASIC.

Errors

- `gml_string` has a numeric value: `Type mismatch` .
- `gml_string` has errors in the GML: `Illegal function call` .
- A variable referenced in the GML string is of incorrect type: `Type mismatch` .

EDIT

```
EDIT {line_number|.}
```

Displays the specified program line with the cursor positioned for editing. `line_number` must be a line that exists in the program, or a period (.) to indicate the last line stored.

Errors

- No line_number is specified: `Undefined line number`.
- More characters are written after the line number: `Illegal function call`.
- `line_number` is not in `[0−65529]`: `Illegal function call`.
- The specified line number does not exist: `Undefined line number`.

ELSE

```
ELSE [anything]
```

Unless part of an `IF` statement on the same line, anything after `ELSE` is ignored in the same way as after `'` or `:REM`. No colon `:` preceding the `ELSE` statement is necessary. See `IF` for normal usage.

END

```
END
```

Closes all files, stops program execution and returns control to the user. No message is printed. It is possible to resume execution at the next statement using `CONT`.

ENVIRON

```
ENVIRON command_string
```

Sets a shell environment variable.

Parameters

`command_string` is a string expression of one of the following forms:

```
"VARIABLE=VALUE"
```

to set `VARIABLE` to `VALUE` ;

```
"VARIABLE="
```

to unset `VARIABLE` .

Errors

- `command_string` has a numeric value: `Type mismatch` .
- `command_string` is not of the required form: `Illegal function call` .

ERASE

```
ERASE array_0 [, array_1] ...
```

De-allocates arrays. The data stored in the arrays is lost.

Parameters

- `array_0`, `array_1 ...` are names of existing arrays. The names must be specified without brackets.

Errors

- No array names are given: `Syntax error` .
- `array_0`, `array_1 ...` do not exist: `Illegal function call` .
- If an error occurs, the arrays named before the error occurred are erased.

```
ERROR error_number
```

Raises the error with number `error_number` .

Parameters

- `error_number` is an expression with a numeric value.

Errors

- `error_number` has a string value: `Type mismatch` .
- `error_number` is not in `[-32768–32767]` : `Overflow` .
- `error_number` is not in `[1–255]` : `Illegal function call` .

FIELD

```
FIELD [#] file_number [, width_0 AS name_0 [, width_1 AS name_1] ...]
```

Assigns variables to the random-access record buffer. The record buffer is a region of memory of length defined by the `OPEN` statement; the default record length is 128 bytes. The `FIELD` statement assigns a portion of this region to one or more fixed-length string variables, so that the value of these strings is whatever happens to be in the record buffer at that location.

Notes

- A `FIELD` statement without any variables specified has no effect.
- Another `FIELD` statement on the same file will specify an alternative mapping of the same file buffer; all mappings will be in effect simultaneously.
- A subsequent assignment or `LET` or `MID$` statement on `name_0 , name_1 ...` will dis- associate the string variable from the field buffer.
- Use `LSET` , `RSET` or `MID$` to copy values into a `FIELD` buffer.
- Use `GET` to read values from the file into the field buffer, changing the variables.
- Use `PUT` to write the field buffer to the file.

Parameters

- `file_number` is a numeric expression that yields the number of an open random-access file. The `#` is optional and has no effect.
- `width_0`, `width_1`, `...` are numeric expressions giving the length of the string variables
- `name_0 , name_1 ...` are string variables.

Errors

- `file_number` is not in `[0–255]` : `Illegal function call` .
- `file_number` is not the number of an open file: `Bad file number` .
- `file_number` is open under a mode other than `RANDOM` : `Bad file mode` .
- The statement ends in a comma: `Missing operand` .
- No file number is specified: `Missing operand` .
- The lengths in a `FIELD` statement add up to a number larger than the record length of the field buffer: `Field overflow` .
- `name_0 , name_1 ...` specify a non-string variable: `Type mismatch` .

FILES

```
FILES [filter_spec]
```

Displays the files fitting the specified filter in the specified directory on a disk device. If `filter_spec` is not specified, displays all files in the current working directory.

Parameters

- `filter_spec` is a string expression that is much like a <u>file specification</u>, but optionally allows the file name part to contain *wildcards*.

Notes

- The filename filter may contain the following wildcards:

`?`	Matches any legal file name character.
`*`	Matches any series of legal file name characters.

- The filter will only match <u>MS-DOS style filenames</u>.
- Matched character series do not stretch across directory separators `\` or extension separators `.`. To match all files with all extensions, use `*.*`.
- Alternatively, if all files in a specified directory are required, end the directory name with a backslash `\`.

Errors

- `filter_spec` has a numeric value: `Type mismatch`.
- `filter_spec` is the empty string: `Bad file name`.
- The specified filter does not match any files: `File not found`.

FOR

```
FOR loop_var = start TO stop [STEP step]
```

Initiates a `FOR–NEXT` loop.

Initially, `loop_var` is set to `start`. Then, the statements between the `FOR` statement and the `NEXT` statement are executed and `loop_var` is incremented by `step` (if `step` is not specified, by 1). This is repeated until `loop_var` has become greater than `stop`. Execution then continues at the statement following `NEXT`. The value of `loop_var` equals `stop+step` after the loop.

`start`, `stop` and `step` are evaluated only once and the resulting values are used throughout the loop.

Parameters

- `loop_var` is an integer or single-precision variable.
- `start`, `stop` and `step` are numeric expressions.

Errors

- No `NEXT` statement is found to match the `FOR` statement: `FOR without NEXT` occurs at the `FOR` statement.
- `loop_var` is a string variable or `start`, `stop`, or `end` has a string value: `Type mismatch`.
- `loop_var` is a double-precision variable: `Type mismatch`.
- `loop_var` is an array element: `Syntax error`.
- `loop_var` is an integer variable and a `start`, `stop` or `step` is outside the range `[-32768, 32767]`: `Overflow`.

GET (files)

```
GET [#] file_number [, record_number]
```

Read a record from the random-access file `file_number` at position `record_number`. The record can be accessed through the `FIELD` variables or through `INPUT$` , `INPUT` or `LINE INPUT` .

Parameters

- `file_number` is a numeric expression that yields the number of an open random-access file. The `#` is optional and has no effect.
- `record_number` is a numeric expression in `[1–33554432]` (`2^25`), and is interpreted as the record number.

Notes

- If the record number is beyond the end of the file, the file buffer is filled with null bytes.
- The record number is stored as single-precision; this precision is not high enough to distinguish single records near the maximum value of `2^25` .

Errors

- `record_number` is not in `[1–33554432]` : `Bad record number` .
- `file_number` is not in `[0–255]` : `Illegal function call` .
- `file_number` is not the number of an open file: `Bad file mode` .
- `file_number` is open under a mode other than `RANDOM` : `Bad file mode` .
- `file_number` is not specified: `Missing operand` .

`GET` (communications)

```
GET [#] com_file_number [, number_bytes]
```

Read `number_bytes` bytes from the communications buffer opened under file number `com_file_number`. The record can be accessed through the `FIELD` variables or through `INPUT$`, `INPUT` or `LINE INPUT`.

Parameters

- `file_number` is a numeric expression that yields the number of a file open to a `COM` device. The `#` is optional and has no effect.
- `number_bytes` is a numeric expression between `1` and the `COM` buffer length, inclusive.

Notes

- If `bytes` is `32768` or greater, GW-BASIC hangs. This functionality is not implemented in PC-BASIC.
- In GW-BASIC, `Device I/O error` is raised for overrun error, framing error, and break interrupt. `Device fault` is raised if DSR is lost during I/O. `Parity error` is raised if parity is enabled and incorrect parity is encountered. This is according to the manual; it is untested.

Errors

- `bytes` is less than 1: `Bad record number`
- `bytes` is less than `32768` and greater than the `COM` buffer length: `Illegal function call`.
- `com_file_number` is not specified: `Missing operand`.
- `com_file_number` is not in `[0—255]`: `Illegal function call`.
- `com_file_number` is not the number of an open file: `Bad file number`.
- If the serial input buffer is full, i.e. `LOF(com_file_number) = 0`, and `LOC(com_file_number) = 255`: `Communication buffer overflow`
- If the carrier drops during `GET`, hangs until the `Ctrl`+`Break` key is pressed.

GET **(graphics)**

```
GET (x0, y0) - [STEP] (x1, y1), array_name
```

Stores a rectangular area of the graphics screen in an array. The area stored is a rectangle parallel to the screen edges, bounded by the top-left and bottom-right coordinates `x0`, `y0` and `x1`, `y1`. If `STEP` is specified, `x1`, `y1` is an offset from `x0`, `y0`. The area is such that these corner points are inside it.

The image stored in the array can then be put on the screen using PUT. For the purposes of `GET`, any array is considered a string of bytes. The byte size of an array can be calculated as `number_elements * byte_size` with `byte_size` equal to `2` for integers (`%`), 4 for single (`!`) and 8 for double (`#`). Array byte size for string is 3, but string arrays are not allowed in `GET`. For calculating the number of elements, keep in mind that `OPTION BASE 0` is the default; in which case an array with maximum index 10 has 11 elements. This works through in multidimensional arrays.

The array format is as follows:

Byte	Contains
0, 1	Number of x pixels, unsigned int. In `SCREEN 1`, this value is doubled.
2, 3	Number of y pixels, unsigned int.
4—	Pixel data. Data is arranged in 2-byte words. The first 16-bit word holds the bit 0 of the first 16 pixels on the top row. The second word holds the second bit, etc. Data is word-aligned at the end of each row. Thus, in a screen mode with 4 bits per pixel, the first row takes at least 8 bytes (4 words), even if it consists of only one pixel. The number of bits per pixel depends on the `SCREEN` mode.

Parameters

- `array_name` is the name of a numeric array dimensioned with enough space to store the area.
- `x0`, `y0`, `x1`, `y1` are numeric expressions.

Notes

- In PCjr/Tandy mode, in `SCREEN 6`, `GET` stores an area of *twice the width* of the specified rectangle.

Errors

- The array does not exist: `Illegal function call`.
- `array_name` refers to a string array: `Type mismatch`.
- The area is too large for the array: `Illegal function call`.

- `x0` , ... `y1` are string expressions: `Type mismatch` .
- `x0` , ... `y1` are not in `[-32768—32767]` : `Overflow` .
- `x0` , ... `y1` are outside the current `VIEW` or `WINDOW` : `Illegal function call`

GOSUB

```
GO[ ]SUB line_number [anything]
```

Jumps to a subroutine at `line_number` . The next `RETURN` statement jumps back to the statement after `GOSUB` . Anything after `line_number` until the end of the statement is ignored. If executed from a direct line, `GOSUB` runs the subroutine and the following `RETURN` returns execution to the direct line.

Parameters
- `line_number` is an existing line number literal.
- Further characters on the line are ignored until end of statement.

Notes
- If no `RETURN` is encountered, no problem.
- One optional space is allowed between `GO` and `SUB` ; it will not be retained in the program.

Errors
- If `line_number` does not exist: `Undefined line number` .
- If `line_number` is greater than `65529` , only the first 4 characters are read (e.g. `6553`)

```
GO[ ]TO line_number [anything]
```

Jumps to `line_number` . Anything after `line_number` until the end of the statement is ignored. If executed from a direct line, `GOTO` starts execution of the program at the specified line.

Parameters

- `line_number` is an existing line number literal.
- Further characters on the line are ignored until end of statement.

Notes

- Any number of optional spaces is allowed between `GO` and `TO` , but they will not be retained in the program.
- If `line_number` is greater than `65529` , only the first 4 characters are read (e.g. `GOTO 65530` is executed as `GOTO 6553`)

Errors

- `line_number` does not exist: `Undefined line number` .

IF

```
IF truth_value [,] {THEN|GOTO} [compound_statement_true|line_number_true [anything]]
    [ELSE [compound_statement_false|line_number_false [anything]]]
```

If `truth_value` is non-zero, executes `compound_statement_true` or jumps to `line_number_true` . If it is zero, executes `compound_statement_false` or jumps to `line_number_false` .

Parameters

- `truth_value` is a numeric expression.
- `line_number_false` and `line_number_true` are existing line numbers.
- `compound_statement_false` and `compound_statement_true` are compound statements, consisting of at least one statement, optionally followed by further statements separated by colons `:` . The compound statements may contain nested `IF—THEN—ELSE` statements.

Notes

- The comma is optional and ignored.
- `ELSE` clauses are optional; they are bound to the innermost free `IF` statement if nested. Additional `ELSE` clauses that have no matching `IF` are ignored.
- All clauses must be on the same program line.
- `THEN` and `GOTO` are interchangeable; which one is chosen is independent of whether a statement or a line number is given. `GOTO PRINT 1` is fine.
- As in `GOTO` , anything after the line number is ignored.

Errors

- If `truth_value` has a string value: `Type mismatch` .
- `truth_value` equals `0` and `line_number_false` is a non-existing line number, or `truth_value` is nonzero and `line_number_true` is a non-existing line number: `Undefined line number` .

`INPUT` **(console)**

`INPUT` [;] [*prompt* {;|,}] *var_0* [, *var_1*] `...`

Prints `prompt` to the screen and waits for the user to input values for the specified variables. The semicolon before the prompt, if present, stops a newline from being printed after the values have been entered. If the prompt is followed by a semicolon, it is printed with a trailing `?` . If the prompt is followed by a comma, no question mark is added.

Parameters

- `prompt` is a string literal.
- `var_0`, `var_1`, `...` are variable names or fully indexed array elements.

Notes

- Values entered must be separated by commas. Leading and trailing whitespace is discarded.
- String values can be entered with or without double quotes (`"`).
- If a string with a comma, leading or trailing whitespace is needed, quotes are the only way to enter it.
- Between a closing quote and the comma at the end of the entry, only white- space is allowed.
- If quotes are needed in the string itself, the first character must be neither a quote nor whitespace. It is not possible to enter a string that starts with a quote through `INPUT` .
- If a given `var_n` is a numeric variable, the value entered must be number literal.
- Characters beyond the 255th character of the screen line are discarded.
- If user input is interrupted by `Ctrl` + `Break` , `CONT` will re-execute the `INPUT` statement.

Errors

- If the value entered for a numeric variable is not a valid numeric literal, or the number of values entered does not match the number of variables in the statement, `?Redo from start` is printed and all values must be entered again.
- A `Syntax error` that is caused after the prompt is printed is only raised after the value shave been entered. No values are stored.

`INPUT` **(files)**

```
INPUT # file_num, var_0 [, var_1] ...
```

Reads string or numeric variables from a text file or the `FIELD` buffer of a random access file.

Parameters

- `file_num` is the number of a file open in `INPUT` mode or a random-access file open in `RANDOM` mode.
- `var_0`, `var_1`, `...` are variable names or fully indexed array elements.

Notes

- The `#` is mandatory. There may or may not be whitespace between `INPUT` and `#`.
- String values can be entered with or without double quotes (`"`).
- Numeric values are terminated by ` `, `LF`, `CR`, `,`.
- Unquoted strings are terminated by `LF`, `CR`, `,`.
- Quoted strings are terminated by the closing quote.
- Any entry is terminated by `EOF` character or its 255th character.
- Leading and trailing whitespace is discarded.
- If the entry cannot be converted to the requested type, a zero value is returned.
- If `file_num` is open to `KYBD:`, `INPUT#` reads from the keyboard until a return or comma is encountered (as in a file). Arrow keys and delete are passed as their control characters (*not* scancodes!) preceded by `CHR$(&hFF)`.

Errors

- Input is requested after the end of a text file has been reached or an `EOF` character has been encountered: `Input past end`.
- The last character of the field buffer is read: `Field overflow`.
- `file_num` has a string value: `Type mismatch`.
- `file_num` is greater than `32767`: `Overflow`.
- `file_num` is less than zero: `Illegal function call`.
- `file_num` is not an open file: `Bad file number`.
- `file_num` is not open for `INPUT` or `RANDOM`: `Bad file mode`.
- `file_num` is open to a `COM` port and this is the first `INPUT`, `LINE INPUT` or `INPUT$` call on that port since the buffer has filled up completely (i.e. `LOF(file_num)` has become zero): `Communication buffer overflow`.

```
IOCTL
```

```
IOCTL [#] file_num, control_string
```

Raises `Illegal function call` .

Notes

- In GW-BASIC, `IOCTL` sends a control string to a device.
- This statement is not implemented in PC-BASIC.

Errors

- `file_num` has a string value: `Type mismatch` .
- `file_num` is not in `[-32768—32767]` : `Overflow` .
- `file_num` is not an open file: `Bad file number` .
- Otherwise: `Illegal function call`

`KEY` (macro list)

```
KEY {ON|OFF|LIST}
```

Turns the list of function-key macros on the bottom of the screen `ON` or `OFF` . If `LIST` is specified, prints a list of the 10 (or 12 with `syntax=tandy`) function keys with the function-key macros defined for those keys to the console.

Most characters are represented by their symbol equivalent in the current codepage. However, some characters get a different representation, which is a symbolic representation of their effect as control characters on the screen.

Code point	Replacement	Usual glyph
&h07	&h0E	♫
&h08	&hFE	■
&h09	&h1A	→
&h0A	&h1B	←
&h0B	&h7F	⌂
&h0C	&h16	▬
&h0D	&h1B	←
&h1C	&h10	▶
&h1D	&h11	◀
&h1E	&h18	↑
&h1F	&h19	↓

`KEY` (macro definition)

```
KEY key_id, string_value
```

Defines the string macro for function key `key_id`. Only the first 15 characters of `string_value` are stored.

Parameters

- `key_id` is a numeric expression in the range `[1–10]` (or `[1–12]` when `syntax=tandy`).
- `string_value` is a string expression.

Notes

- If `key_id` is not in the prescribed range, the statement is interpreted as an event-trapping `KEY` statement.
- If `string_value` is the empty string or the first character of `string_value` is `CHR$(0)`, the function key macro is switched off and subsequent catching of the associated function key with `INKEY$` is enabled.

Errors

- `key_id` is not in `[-32768–32767]`: `Overflow`.
- `key_id` is not in `[1–255]`: `Illegal function call`.
- `key_id` has a string value: `Type mismatch`.

`KEY` (event switch)

```
KEY (key_id) {ON|OFF|STOP}
```

Controls event trapping of the key with identifier `key_id` . Event trapping is switched `ON` or `OFF` . `STOP` suspends event trapping until a `KEY() ON` is executed. Up to one event can be triggered during suspension, provided that event handling was switched on prior to suspension. The event triggered during suspension is handled immediately after the next `KEY() ON` statement.

Parameters

`key_id` is a numeric expression in `[1–20]` . Keys are:

1	F1
2	F2
3	F3
4	F4
5	F5
6	F6
7	F7
8	F8
9	F9
10	F10
11	↑
12	←
13	→
14	↓

Keys 15 to 20 are defined using the event trapping `KEY` definition statement.

Notes
- With `syntax=tandy` , key 11 is `F11` and key 12 is `F12` . Pre-defined keys 11—14 shift to 13—16.

Errors
- `key_id` is not in `[-32768–32767]` : `Overflow` .
- `key_id` is not in `[0–20]` : `Illegal function call` .
- `key_id` has a string value: `Type mismatch` .

`KEY` (event definition)

```
KEY key_id, two_char_string
```

Defines the key to trap for `key_id`.

Parameters

- `key_id` is a numeric expression in `[15—20]` (or `[17—20]` when `syntax=tandy`).
- `two_char_string` is a string expression of length 2. The first character is interpreted as a modifier while the second character is interpreted as a scancode. The modifier character is a bitwise `OR` combination of the following flags:

`CHR$(&h80)`	Extended
`CHR$(&h40)`	`Caps Lock`
`CHR$(&h20)`	`Num Lock`
`CHR$(&h08)`	`Alt`
`CHR$(&h04)`	`Ctrl`
`CHR$(&h02)`	`Shift` (either side)
`CHR$(&h01)`	`Shift` (either side)

For the unmodified key, the modifier character is `CHR$(0)`.

Notes

- If `key_id` is not in the prescribed range, no error is raised; such values are ignored. In GW-BASIC strange things can happen in this case: screen anomalies and crashes suggestive of unintended memory access.
- If `key_id` is in `[1—10]` (or `[1—12]` when `syntax=tandy`), the statement is interpreted as a function-key macro definition.

Errors

- `key_id` is not in `[-32768—32767]`: `Overflow`.
- `key_id` is not in `[1—255]`: `Illegal function call`.
- `key_id` has a string value: `Type mismatch`.
- `two_char_string` is longer than two: `Illegal function call`.
- `two_char_string` has a numeric value: `Type mismatch`.

KILL

```
KILL filter_spec
```

Deletes one or more files on a disk device.

Parameters

- The string expression `filter_spec` is a valid <u>file specification</u> indicating the files to delete. Wildcards are allowed. See <u>FILES</u> for a description of wildcards.

Notes

- Be very careful with the use of wildcards in this statement: the DOS matching rules may not be the same as what is usual on your operating system, which could result in unexpected files being deleted.
- This statement may not delete hidden file and files that do not have short names which are legal DOS names. However, this behaviour is not guaranteed so you must not depend on it.

Errors

- `filter_spec` is a numeric value: `Type mismatch` .
- A file with a base name equal to that of a file matching `filter_spec` is open: `File already open`
- No file matches `filter_spec` : `File not found`
- The user has no write permission: `Permission denied`
- If a syntax error occurs after the closing quote, the file is removed anyway.

LCOPY

```
LCOPY [num]
```

Does nothing.

Parameters

- `num` is a numeric expression in `[0–255]`.

Notes

- This statement does nothing in GW-BASIC. Presumably, it is left over from a statement in older versions of MS Basic that would copy the screen to the printer.

Errors

- `num` is not in `[-32768–32767]`: `Overflow`.
- `num` is not in `[0–255]`: `Illegal function call`.
- `num` has a string value: `Type mismatch`.

LET

```
[LET] name = expression
```

Assigns the value of `expression` to the variable or array element `name`.

Parameters

- `name` is a variable that may or may not already exist.
- The type of `expression` matches that of `name`: that is, all numeric types can be assigned to each other but strings can only be assigned to strings.

Errors

- `name` and `expression` are not of matching types: `Type mismatch`.

LINE [[**STEP**] (*x0*, *y0*)] - [**STEP**] (*x1*, *y1*) [, [*attr*] [, [**B** [**F**]] [, *pattern*]]]

Draws a line or a box in graphics mode. If `B` is not specified, a line is drawn from `(x0, y0)` to `(x1, y1)`, endpoints inclusive. If `B` is specified, a rectangle is drawn with sides parallel to the screen and two opposing corners specified by `(x0, y0)` and `(x1, y1)`. If the starting point is not given, the current graphics position is used as a staring point. If `STEP` is specified, `(x0, y0)` is an offset from the current position and `(x1, y1)` is an offset from `(x0, y0)`. `LINE` moves the current graphics position to the last given endpoint. If `F` is specified with `B`, the rectangle is filled with the specified attribute. `F` and `B` may be separated by zero or more spaces.

Parameters

- `attr` is a numeric expression in `[0–255]`, which specifies the colour attribute of the line. If it is not given, the current attribute is used.
- `pattern` is a numeric expression in `[-32768–32767]`. This is interpreted as a 16-bit binary pattern mask applied to consecutive pixels in the line: a 1 bit indicates a pixel plotted; a 0 bit indicates a pixel left untouched. The pattern starts at the most significant bit, which is applied to the topmost endpoint. If a box is drawn, the pattern is applied in the following counter-intuitive sequence: `(x1, y1)−(x0, y1)`, `(x1, y0)−(x0, y0)`, then `(x1, y0)−(x1, y1)`, `(x0, y0)−(x0, y1)` if `y0<y1` and `y0`, `y1` reversed if `y1<y0`. When drawing a filled box, `LINE` ignores the pattern.

Notes

- If a coordinate is outside the screen boundary, it is replaced with -1 (if less than 0) or the screen dimension (if larger than the screen dimension).

Errors

- The statement ends in a comma and it is the first or third: `Missing operand`. If it is the second: `Syntax error`.
- Any of the coordinates is not in `[-32768–32767]`: `Overflow`.
- Any of the parameters has a string value: `Type mismatch`.

`LINE INPUT` (console)

```
LINE INPUT [;] [prompt_literal {;|,}] string_name
```

Displays the prompt given in `prompt_literal` and reads user input from the keyboard, storing it into the variable `string_name`. All input is read until `Enter` is pressed; the first 255 characters are stored. If the `;` is given right after `LINE INPUT`, the `Enter` ending user input is not echoed to the screen.

Parameters

- `prompt_literal` is a string literal. It makes no difference whether it is followed by a comma or a semicolon.
- `string_name` is a string variable or array element.

Notes

- If user input is interrupted by `Ctrl`+`Break`, `CONT` will re-execute the `LINE INPUT` statement.
- Unlike `INPUT`, `LINE INPUT` does not end the prompt with `?`.

`LINE INPUT` (files)

`LINE INPUT #` *file_num*, *string_name*

Reads string or numeric variables from a text file or the `FIELD` buffer of a random access file. All input is read until `Enter` is pressed; the first 255 characters are stored. *file_num* must be the number of a file open in `INPUT` mode or a random-access file open in `RANDOM` mode.

Parameters

- *string_name* is a string variable or array element.

Notes

- The `#` is mandatory. There may or may not be whitespace between `INPUT` and `#`.
- Input is only terminated by a `CR`.
- If *file_num* is open to `KYBD:`, `LINE INPUT#` reads from the keyboard until a return or comma is encountered (as in a file). Arrow keys and delete are passed as their control characters (*not* scancodes!) preceded by `CHR$(&hFF)`.

Errors

- Input is requested after the end of a text file has been reached or an `EOF` char has been encountered: `Input past end`.
- The last character of the field buffer is read: `Field overflow`.
- *file_num* is not an open file: `Bad file number`.
- *file_num* is less than zero: `Illegal function call`.
- *file_num* is not in `[-32768–32767]`: `Overflow`.
- *file_num* is not open for `INPUT` or `RANDOM`: `Bad file mode`.
- *file_num* has a string value: `Type mismatch`.
- *file_num* is open to a `COM` port and this is the first `INPUT`, `LINE INPUT` or `INPUT$` call on that port since the buffer has filled up completely (i.e. `LOF(`*file_num*`)` has become zero): `Communication buffer overflow`.

LIST

```
LIST [line_number_0|.] [-[line_number_1|.]] [, file_spec [anything]]
```

Prints the program to the screen or a file, starting with `line_number_0` up to and including `line_number_1`. Also stops program execution and returns control to the user. If the `LIST` statement ends with a file specification, anything further is ignored. In all cases, any further statements in a compound after `LIST` will be ignored, both in a program and in direct mode.

When listing to the screen, the same control characters are recognised as in the `PRINT` statement.

Notes

- In GW-BASIC 3.23, `LIST` will not show line numbers `65531` — `65535` inclusive. By default, PC-BASIC's `LIST` does show these lines. However, showing them can be disabled with the option `hide-listing=65530`.

Parameters

- `line_number_0` and `line_number_1` are line numbers in the range `[0–65529]` or a `.` to indicate the last line edited. The line numbers do not need to exist; they specify a range. If the range is empty, nothing is printed.
- The string expression `file_spec` is a valid file specification indicating the file to list to. If this file already exists, it will be overwritten.

Errors

- A line number is greater than `65529`: `Syntax error`.
- `file_spec` has a numeric value: `Type mismatch`.
- `file_spec` ends in a colon but is not a device name or drive letter: `Bad file number`.
- `file_spec` contains disallowed characters: `Bad file number` (on `CAS1:`); `Bad file name` (on disk devices).

LLIST

`LLIST` [*line_number_0*|.] [-[*line_number_1*|.]]

Prints the program to the screen, starting with `line_number_0` up to and including `line_number_1`. Also stops program execution and returns control to the user. Any further statements on a line after `LLIST` will be ignored, both in a program and in direct mode.

Notes

- In GW-BASIC 3.23, `LLIST` will not show line numbers `65531` — `65535` inclusive. By default, PC-BASIC's `LLIST` does show these lines. However, showing them can be disabled with the option `hide-listing=65530`.

Parameters

- `line_number_0` and `line_number_1` are line numbers in the range `[0-65529]`. or a `.` to indicate the last line edited. The line numbers do not need to exist; they specify a range. If the range is empty, nothing is printed.

Errors

- A line number is greater than `65529`: `Syntax error`.

LOAD

```
LOAD file_spec [, R]
```

Loads the program stored in a file into memory. Existing variables will be cleared and any program in memory will be erased. `LOAD` implies a `CLEAR` .

If `,R` is specified, keeps all data files open and runs the specified file.

Parameters

- The string expression `file_spec` is a valid <u>file specification</u> indicating the file to read the program from.

Errors

- `file_spec` has a numeric value: `Type mismatch` .
- `file_spec` contains disallowed characters: `Bad file number` (on `CAS1:`); `Bad file name` (on disk devices).
- The file specified in `file_spec` cannot be found: `File not found` .
- A loaded text file contains lines without line numbers: `Direct statement in file` .
- A loaded text file contains lines longer than 255 characters: `Line buffer overflow` . Attempting to load a text file that has `LF` rather than `CR LF` line endings may cause this error.

`LOCATE` [*row*] [, [*col*] [, [*cursor_visible*] [, [*start_line*] [, [*stop_line*] [,]]]]]

Positions the cursor at `row` , `col` on the screen and changes the cursor shape and visibility. `cursor_visible` may be 0 or 1. If `cursor_visible` is 0, it makes the cursor invisible; if it is 1, makes the cursor visible. This works only while a program is running. The cursor shape is adjusted within a character cell to start from `start_line` and end on `end_line` where `start_line` and `end_line` are in `[0–31]` . If `start_line` or `end_line` is greater than the character cell height (15), substitute 15.

Notes
- On emulated VGA cards, the cursor shape parameters are interpreted in a complicated way that is intended to maintain functional compatibility with CGA.
- In GW-BASIC, cursor shape is preserved after pressing `Ins` twice. The insert-mode cursor is different from the usual half-block. In PC-BASIC, insert mode resets the cursor shape to default.
- Cursor shape and visibility options have no effect in graphics mode.
- Locate accepts a 5th comma at the end, which is ignored.

Errors
- Any parameter has a string value: `Type mismatch` .
- Any parameter is not in `[-32768–32767]` : `Overflow` .
- `row` is outside the current view area: `Illegal function call` .
- `col` is greater than the current width: `Illegal function call` .
- `cursor_visible` is not in `[0, 1]` (`[0–255]` on Tandy/PCjr): `Illegal function call` .

LOCK

```
LOCK [#] file_number [, record_0]
```

```
LOCK [#] file_number, [record_0] TO record_1
```

Locks a file or part of a file against access by other users. On a `RANDOM` file, `record_0` is the first record locked and `record_1` is the last record locked. On any other kind of file `record_0` and `record_1` have no effect. If `record_0` is not specified, it is assumed to be 1. If no records are specified, the whole file is locked.

Parameters

- `file_number` is a numeric expression in `[0–255]`.
- `record_0` and `record_1` are numeric expressions in `[1–2^25-2]`.

Notes

- In GW-BASIC under MS-DOS, the `LOCK` command requires `SHARE.EXE` to be loaded. The maximum number of locks is specified in the MS-DOS `SHARE` command. If `SHARE` has not been activated or all locks are used, `LOCK` raises `Permission denied`. PC-BASIC behaves as if `SHARE` has been activated with unlimited locks.
- If `file_number` is open for `RANDOM`, `LOCK` and `UNLOCK` statements must match in terms of `record_0` and `record_1`. An non-matching `UNLOCK` will raise `Permission denied`.
- To check if another open file is the same file, PC-BASIC only looks at the base name of the file, i.e. its DOS name without directories. As a consequence, if a file `"test.txt"` is open and locked, an attempt to lock a file `"dir\test.txt"` will fail, even if these are different files. Conversely, if two file names are different but point to the same file in the file system (for example due to file system links), then these will be considered as different files by BASIC.

Errors

- Any parameter has a string value: `Type mismatch`.
- `file_num` is not in `[-32768–32767]`: `Overflow`.
- `file_num` is not in `[0–255]`: `Illegal function call`.
- `file_num` is not an open file: `Bad file number`.
- `LOCK` (part of) a file with the same name as a file already locked: `Permission denied`.
- `record_0` or `record_1` is not in `[1–2^25-2]`: `Bad record number`.

LPRINT

See `PRINT` .

LSET

```
LSET string_name = expression
```

Copies a string value into an existing string variable or array element. The value will be left-justified and any remaining characters are replaced by spaces.

Parameters
- `string_name` is a string variable or array element.
- `expression` is a string expression.

Notes
- If `expression` has a value that is longer than the length of the target variable, it is truncated at the tail to the length of the target variable.
- If `string_name` has not been allocated before, this statement has no effect.
- Use `LSET` , `RSET` or `MID$` to copy values into a `FIELD` buffer.
- If `LET` is used on a `FIELD` variable instead of `L|RSET` , the variable is detached from the field and a new, normal string variable is allocated.

Errors
- `string_name` is not a string variable: `Type mismatch` .
- `expression` does not have a string value: `Type mismatch` .

MERGE

```
MERGE file_spec
```

Overlays the lines of a program from a plain-text program file into the existing program. The loaded lines overwrite existing lines if they have the same line number.

Parameters

- The string expression `file_spec` is a valid <u>file specification</u> indicating the file to read the program from.

Errors

- `file_spec` cannot be found: `File not found`.
- `file_spec` contains disallowed characters: `Bad file number` (on `CAS1:`); `Bad file name` (on disk devices).
- `file_spec` was not saved as plain text: `Bad file mode`.
- A loaded text file contains lines without line numbers: `Direct statement in file`.
- A loaded text file contains lines longer than 255 characters: `Line buffer overflow`. Attempting to load a text file that has `LF` rather than `CR LF` line endings may cause this error.

MID$ (statement)

```
MID$(string_name, position [, length]) = substring
```

Replaces part of `string_name` with `substring`.

Parameters

- `string_name` is a valid string variable name.
- `position` is a numeric expression between 1 and the string length, inclusive.
- `length` is a numeric expression in `[0–255]`.

Notes

- No whitespace is allowed between `MID$` and `(`.
- If `substring` is longer than `length`, only the first `length` characters are used.
- If `substring` is shorter than `length`, only `LEN(substring)` characters are replaced.

Errors

- `position` is greater than the length of `string_name`: `Illegal function call`, except if `length` is specified as 0.
- `position` is not in `[1–255]`: `Illegal function call`.
- `length` is not in `[0–255]`: `Illegal function call`.
- `position` or `length` are not in `[-32768–32767]`: `Overflow`.

MKDIR

```
MKDIR dir_spec
```

Creates a new directory on a disk device.

Parameters

- The string expression `dir_spec` is a valid <u>file specification</u> that specifies the path of the new directory on a disk device.

Errors

- `dir_spec` is not a string: `Type mismatch`.
- The parent directory does not exist: `Path not found`.
- The directory name already exists on that path: `Path/File access error`.
- The user has no write permission: `Permission denied`.

MOTOR

`MOTOR` [*num*]

Does nothing.

Parameters

- *num* is a numeric expression in `[0–255]`.

Notes

- In GW-BASIC, this statement turns on the cassette motor if *num* is nonzero or omitted, and turns it off if *num* is zero. This is not implemented in PC-BASIC.

Errors

- *num* has a string value: `Type mismatch`.
- *num* is not in `[-32768–32767]`: `Overflow`.
- *num* is not in `[0–255]`: `Illegal function call`.

NAME

`NAME` *old_name* `AS` *new_name*

Renames the disk file *old_name* into *new_name*.

Parameters

- The string expressions *old_name* and *new_name* are valid <u>file specifications</u> giving the path on a disk device to the old and new filenames, respectively.

Notes

- *new_name* will be modified into <u>all-uppercase 8.3 format</u>.

Errors

- *old_name* or *new_name* have number values: `Type mismatch`.
- *old_name* does not exist: `File not found`.
- A file with a base name equal to that of *old_name* or *new_name* is open: `File already open`.
- *new_name* exists: `File already exists`.

NEW

Stops execution of a program, deletes the program in memory, executes `CLEAR` and `RESTORE` and returns control to the user.

NEXT

NEXT [*var_0* [, *var_1*] ...]

Iterates a `FOR–NEXT` loop: increments the loop variable and jumps to the `FOR` statement. If no variables are specified, next matches the most recent `FOR` statement. Several nested `NEXT` statements can be consolidated into one by using the variable list. If one or more variables are specified, their order must match the order of earlier `FOR` statements.

Parameters

- *var_0*, *var_1*, ... are numeric variables which are loop counters in a `FOR` statement.

Errors

- No `FOR` statement is found to match the `NEXT` statement and variables: `NEXT without FOR`.
- *var_0*, *var_1*, ... are string variables: `NEXT without FOR`.
- The (implicit or explicit) loop variable is an integer variable and is taken outside the range `[-32768, 32767]` when incremented after the final iteration: `Overflow`.

NOISE

```
NOISE source, volume, duration
```

Generates various kinds of noise.

Parameters

- `source` is a numeric expression in `[0−7]` . It indicates the type of noise:

source	type	top of frequency band (Hz)
0	periodic	6991
1	periodic	3495
2	periodic	1747
3	periodic	last tone played on voice 2
0	white noise	6991
1	white noise	3495
2	white noise	1747
3	white noise	last tone played on voice 2

- `volume` is a numeric expression in `[0−15]` .
- `duration` is a numeric expression.

Volume and duration are determined in the same way as for the `SOUND` statement; see there.

Notes

- This statement is only available if `syntax={pcjr|tandy}` is set.

Errors

- `SOUND ON` has not been executed: `Illegal function call` .
- `duration` is not in `[-65535−65535]` : `Illegal function call` .
- `volume` is not in `[0−15]` : `Illegal function call` .
- `source` is not in `[0−7]` : `Illegal function call` .

ON (calculated jump)

```
ON n {GOTO|GOSUB} line_number_0 [, line_number_1] ...
```

Jumps to the `n` th line number specified in the list. If `n` is 0 or greater than the number of line numbers in the list, no jump is performed. If `GOTO` is specified, the jump is unconditional; if `GOSUB` is specified, jumps to a subroutine.

Parameters

- `n` is a numeric expression in `[0–255]`. The expression must not start with the `STRIG`, `PEN`, `PLAY` or `TIMER` function keywords; if you need these functions, the expression must be bracketed.
- `line_number_0, line_number_1, ...` are existing line numbers in the program.

Errors

- `n` has a string value: `Type mismatch`.
- `n` is not in `[-32768–32767]`, `Overflow`.
- `n` is not in `[0–255]`: `Illegal function call`.
- The line number jumped to does not exist: `Undefined line number`.

ON (event trapping)

```
ON {COM(n)|KEY(n)|STRIG(n)|PEN|PLAY(n)|TIMER(x)} GOSUB line_number
```

Defines an event trapping subroutine. The type of event is given by one of the following keywords:

COM(*n*)	The event is triggered if data is present in the input buffer of the COM*n*:. *n* is the port number in [1,2].
KEY(*n*)	The event is triggered if key *n* is pressed. *n* is the key number [1–20] defined in the KEY statement.
STRIG(*n*)	They event is triggered if fire button *n* is pressed. *n* in [0,2,4,6] refer to the two fire triggers on two joysticks.
PEN	The event is triggered if the light pen is on the screen. (In PC-BASIC, the light pen is emulated by default by the right mouse button).
PLAY(*n*)	The event is triggered if there are exactly *n* notes left on the music background queue. *n* is a numeric expression in [1–32].
TIMER(*x*)	The event is triggered every *x* seconds after the TIMER ON statement. *x* is a numeric expression in [1–86400].

Notes

- Event trapping for your chosen event first needs to be enabled using one of the statements: COM(*n*) ON , KEY(*n*) ON , STRIG(*n*) ON , PEN ON , PLAY ON , TIMER ON
- Events are only trapped when a program is running.

Errors

- *n* or *x* has a string value: Type mismatch .
- *n* is not in [-32768–32767] : Overflow .
- *n* or *x* is outside the specified range: Illegal function call .

ON ERROR

```
ON ERROR GOTO {line_number|0}
```

Turns error trapping on or off. When `line_number` is set, any error causes the error handling routine starting at that line number to be called; no message is printed and program execution is not stopped. The error handling routine is ended by a `RESUME` statement. While in an error handling routine, events are paused and error trapping is disabled. After the `RESUME` statement, any triggered events are picked up in the following order: `KEY`, `TIMER`, `PLAY` - the order of the others is unknown. Unlike event trapping, error trapping remains active when no program is running. `ON ERROR GOTO 0` turns off error trapping.

Parameters

- `line_number` is an existing line number in the program.

Notes

- It is not possible to start the error handler at line number `0`.

Errors

- `line_number` does not exist: `Undefined line number`.

OPEN

```
OPEN mode_char, [#] file_num, file_spec [, rec_len]
```

```
OPEN file_spec [FOR {INPUT|OUTPUT|APPEND|RANDOM}] [ACCESS {READ|WRITE|READ WRITE}]
    [SHARED|LOCK {READ|WRITE|READ WRITE}] AS [#] file_num [LEN = rec_len]
```

Opens a data file on a device.

Parameters

- The string expression `file_spec` is a valid file specification.
- `file_num` is a numeric expression in `[1—max_files]`, where `max_files` is the maximum file number (default 3).
- `rec_len` is a numeric expression in `[1—128]`: the record length.
- `mode_char` is a string expression of which the first character is one of `["I", "O", "A", "R"]`.

Access modes

The `FOR` modes or `mode_char` are as follows:

mode_char	FOR	Effect
"I"	INPUT	Opens a text file for reading and positions the file pointer at the start.
"O"	OUTPUT	Truncates a text file at the start and opens it for writing. Any data previously present in the file will be deleted.
"A"	APPEND	Opens a text file for writing at the end of any existing data.
"R"	RANDOM	Opens a file for random access; the file is divided in records of length `rec_len`. If LEN is not specified, the record length defaults to 128. The file contents can be accessed using GET and PUT of the FIELD buffer; the FIELD buffer can be accessed through FIELD variables or through PRINT# and INPUT# statements.

If no `FOR` mode or `mode_char` is specified, the file is opened for `RANDOM`.

If both `FOR` and `ACCESS` are specified, any `ACCESS` mode is allowed for `RANDOM` but for the other modes the access must match as follows:

FOR	default ACCESS	allowed ACCESS
INPUT	READ	READ
OUTPUT	WRITE	WRITE
APPEND	READ WRITE	READ WRITE
RANDOM	READ WRITE	all

Sharing and locks

If neither `SHARED` nor `LOCK` are specified. Inside this process, a file may be opened multiple times for `INPUT` or `RANDOM` but only once for `OUTPUT` or `APPEND`, as long as it is again opened in default mode. It may not be opened in `SHARED` or any `LOCK` modes.

If `SHARED`, `LOCK READ`, `LOCK WRITE` or `LOCK READ WRITE` is specified, whether two `OPEN` statements may access the same file depends on one's `LOCK` status and the other's `ACCESS` status and vice versa. For two `OPEN` statements as follows:

```
OPEN "file" lock_1 AS 1
OPEN "file" ACCESS acc_2 AS 2
```

the following combinations are allowed:

Access allowed		acc_2		
		READ	WRITE	READ WRITE
lock_1	SHARED	yes	yes	yes
	LOCK READ	no	yes	no
	LOCK WRITE	yes	no	no
	LOCK READ WRITE	no	no	no

In GW-BASIC under MS-DOS with `SHARE.EXE` active, these locks should be enforced across a network as well as inside a single BASIC process. Without `SHARED` and `LOCK`, the file is locked exclusively for use by the GW-BASIC process. By contrast, in PC-BASIC, the locks are only implemented internally. Whether other processes may access the file will depend on the host OS.

To check if another open file is the same file, PC-BASIC only looks at the base name of the file, i.e. its DOS name without directories. As a consequence, if a file `"test.txt"` is open and locked, an attempt to lock a file `"dir\test.txt"` will fail, even if these are different files. Conversely, if two file names are different but point to the same file in the file system (for example due to file system links), then these will be considered as different files by BASIC.

File specifications

A *file specification* `file_spec` is a non-empty string expression of the form
`"[device:]parameters"`, where `device` is a PC-BASIC device and the form of the
`parameters` is specific to the type of device. If `device` is omitted, the current device (one of
the disk devices or `CAS1:`) is used.

Disk devices `A:` — `Z:` and `@:`

`parameters` must specify a valid file path of the form `[\][dirname\] ... filename` .

PC-BASIC follows DOS file system conventions. Directory names are separated with
backslashes `\` (even if the host OS separates paths with forward slashes). File and
directory names consist of a 8-character name and 3-character extension. Names are
case insensitive. Permissible characters for both filename and extension are the
printable ASCII characters in the range `&h20` – `&h7E` excluding the characters `" * +
. , / : ; < = > ? \ [ ] |` . Spaces are allowed but leading and trailing spaces are
ignored. The names `AUX` , `CON` , `PRN` and `NUL` are reserved as device aliases and
are not legal names for files or directories on a disk device.

A path starting with a backslash is interpreted as an absolute path, starting at the root
of the specified disk device. Otherwise, the path is interpreted as relative to the
current directory on the specified device. The special directory name `..` refers to the
parent directory of a preceding path, or the parent directory of the current directory if
no path is given. The special directory name `.` refers to the same directory as given
by the preceding path, or the current directory if no preceding path is given.

If the file name provided does not contain any dots, the `LOAD` , `SAVE` , `BLOAD` ,
`BSAVE` , `CHAIN` , `MERGE` , `RUN` , and `LIST` statements append the default extension
`.BAS` . To refer to a file name without an extension, the file specification should end in
a dot `.` . For other statements, appending a dot is allowed but not required.

Compatibility notes

Unlike PC-BASIC, some versions of MS-DOS allow certain characters in the range
`&h7F` – `&hFF` . However, their permissibility and interpretation depends on the console
code page, which may be different from the display code page that affects GW-
BASIC. Depending on its console code page, MS-DOS will replace accented letters
by their unaccented uppercase variant. Some DOS implementations will remove
spaces from filenames; notably, this is the case on DOSBox.

In order to allow access to files whose name on the host system does not conform to
DOS standards while maintaining compatibility with GW-BASIC, PC-BASIC will follow
these steps to match DOS-style file names to host file names:

1. Look for a file with the name as provided. This can be a long file name which may contain non-permissible characters and which will be case sensitive if your file system is.
2. If such a file is not found, it will truncate the name provided to all-uppercase 8.3 format and look for an exact match. The truncated name consists of the first 8 characters before the first dot, followed by the first three characters after the first dot. If the resulting file name contains non-permissible characters, an error will be raised.
3. Look for 8.3 names in mixed case which match the name provided in a case-insensitive way. Such files are searched in lexicographic order. File names longer than 8.3 will not be matched, unless their name is entered exactly. On Windows, the name matched can be a short filename as well as a long filename provided it is of 8.3 length — it may, for example, contain spaces and thus not be a valid Windows short file name.

If the file name provided ends in a single dot and contains no other dots, PC-BASIC will first match the name as provided; if this is not found, it will match the name as provided but without the single dot. The 8.3 format of such a file name will match file names with and without the dot, in lexicographic order.

If no matching file is found for an output file name, a new file will be created with an all-uppercase 8.3 file name.

Cassette device `CAS1:`

`parameters` can be a file name of up to eight characters. Cassette file names are case sensitive, have no path or extension, may be empty and do not need to be unique. They may contain any character in the range `&h20` – `&hFF` . On the cassette device, when called in direct mode, `OPEN` , `CHAIN` , `MERGE` , `LOAD` and `BLOAD` will print a message to the console for each file found while winding the tape. The message consists of the filename followed by a dot and the file type and concluded with a status message. The file type is one of the following:

`A`	Program file in text format
`B`	Program file in tokenised format
`D`	Data file
`M`	`BSAVE` memory image
`P`	Program file in protected format

If the file does not match the file specification and required file type, the status is `Skipped` ; if the file matches, the status is `Found` . When called from a program, these statements do not print messages to the console. If the `device` was specified explicitly, `parameters` may also be empty. In this case the first file of the appropriate type is opened.

Console and parallel devices `SCRN:`, `KYBD:`, and `LPTn:`

These devices do not allow further device parameters.

Serial devices `COMn:`

When opening a `COM` port, the `file_spec` has the form

```
"COMn:[speed[,parity[,data[,stop[,RS][,CS[n]][,DS[n]][,CD[n]][,LF][,PE]]]]]"
```

The first four parameters after the device colon must be given in the order specified but the named parameters can be given in any order. The meaning of the parameters is:

Parameter	Default	Meaning
speed	`300`	Baud (bps) rate for the connection. `speed` is one of `[75, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600]`.
parity	`E`	Parity bit convention. `parity` is one of `[S, M, O, E, N]`.

`parity`	Meaning	Effect
`S`	SPACE	Parity bit always set to 0.
`M`	MARK	Parity bit always set to 1.
`O`	ODD	Parity bit set so that character parity is odd.
`E`	EVEN	Parity bit set so that character parity is even.
`N`	NONE	No parity bit transmitted or received.

Parameter	Default	Meaning
data	`7`	Data bits per byte. `data` must be one of `[4, 5, 6, 7, 8]`. A byte consists of the data bits plus parity bit, if any. Byte size must be in the range `[5–8]`: if `data` is `4`, `parity` must not be `N`; if `data` is `8`, `parity` must be `N`.
stop	`1`	The number of stop bits. `stop` must be `1` or `2`. Default is `2` if `speed` is `75` or `110`; `1` otherwise.
`RS`	`no`	Suppress *Request To Send*.
`CS[n]`	`CS1000`	Set *Clear To Send* timeout to `n` milliseconds. If `n` is `0` or not given, disable CTS check. Default is `CS0` if `RS` is set; `CS1000` otherwise.
`DS[n]`	`DS1000`	Set *Data Set Ready* timeout to `n` milliseconds. If `n` is `0` or not given, disable DSR check.
`CD[n]`	`CD0`	Set *Carrier Detect* timeout to `n` milliseconds. If `n` is `0` or not given, disable CD check.
`LF`	`no`	Send a line feed after each carriage return.
`PE`	`no`	Enable parity checking (This setting is ignored by PC-BASIC).

Notes

- If a `COM` port is opened for `RANDOM`, access is byte-for-byte rather than through `FIELD` records; `PRINT#` and `INPUT#` access the port directly. `rec_len` sets the number of bytes read by the `GET` and `PUT` statements.
- For `INPUT`, `OUTPUT` and `APPEND` modes, `LEN` may be specified but is ignored.
- If I/O is attempted contravening the `FOR` mode specified, the `PRINT` or `INPUT`

statement will raise `Bad file mode` .

- If `RANDOM` I/O is attempted contravening the `ACCESS` mode specified, the `PUT` or `GET` statement will raise `Path/File access error` .
- The `#` is optional and has no effect.

Errors

- *file_spec* is empty or a non-existent device: `Bad file number` .
- `FOR APPEND ACCESS WRITE` is specified: `Path/File access error` .
- `FOR` and `ACCESS` mismatch in other ways: `Syntax error` .
- The `COM:` *file_spec* parameters do not follow the specification: `Bad file name` .
- The `CAS1:` *file_spec* contains disallowed characters: `Bad file number` .
- A file with the same name is already open for `OUTPUT` or `APPEND` : `File already open` . This is only raised for `COMn:` , `CASn:` and disk devices.
- *rec_len* or *file_num* have string values: `Type mismatch` .
- *file_spec* or *mode_char* have number values: `Type mismatch` .
- *file_num* is not in `[-32768–32767]` : `Overflow` .
- *file_num* is not in `[0–255]` : `Illegal function call` .
- *file_num* is not in `[1–max_files]` : `Bad file number` .
- *rec_len* is not in `[-32768–32767]` : `Overflow` .
- *rec_len* is not in `[1–128]` : `Illegal function call` .
- *mode_char* is empty or the first character is not in `["I", "O", "A", "R"]` : `Bad file mode` .

OPTION BASE

`OPTION BASE` *n*

Sets the starting index of all arrays to `n` .

Parameters

- `n` is a literal digit `0` or `1` . Expressions are not allowed.

Notes

- If `OPTION BASE` has not been called, the first array allocation defaults to starting index 0.

Errors

- `n` is not a digit `0` or `1` : `Syntax error` .
- `OPTION BASE 1` is called but an array has already been allocated before: `Duplicate definition` .
- `OPTION BASE` is called more than once with different starting index: `Duplicate definition` .

OUT

```
OUT port, value
```

Sends a byte to an emulated machine port.

The following machine ports are emulated in PC-BASIC:

port	Effect
&h201	resets the game port (joystick port)
&h3C5	sets the write bitmask for SCREEN 7, 8, 9 colour planes. *bitmask = 2 ^ value*.
&h3CF	sets the read colour plane to *value*.
&h3D8	if *value* = &h1A, enable composite colorburst.
	if *value* = &h1E, disable composite colorburst.
	Requires video={cga, tandy, pcjr}.

Notes

- Only a limited number of machine ports are emulated.
- In GW-BASIC under MS-DOS, the sequence needed to set the colour plane mask is:

```
OUT &h3C4, 2
OUT &h3C5, 2 ^ plane
```

The sequence needed to set the colour plane is:

```
OUT &h3CE, 4
OUT &h3CF, plane
```

The initial OUT statements currently have no effect in PC-BASIC.

Parameters

- *port* is a numeric expression in [-32768—65535].
- *value* is a numeric expression in [0—255].

Errors

- *port* or *value* has a string value: Type mismatch.
- *port* is not in [-32768—65535]: Overflow.
- *value* is not in [-32768—32767]: Overflow.
- *value* is not in [0—255]: Illegal function call.

```
PAINT [STEP] (x, y) [, attrib [, border [, background]]]
```

Flood-fills the screen with a colour or pattern, starting from the given seed point.

Parameters

- `x` , `y` are numeric expressions in the range `[-32768–32767]` If `STEP` is specified, `x` `y` are offsets from the current position. If the seed point is outside the visible screen area, no flood fill is performed.
- `attrib` is an expression that specifies the fill attribute or pattern. If not specified, the current foreground attribute is used.
- If `attrib` has a number value, it must be in `[0–255]` ; it specifies the colour attribute used to fill.
- If `attrib` has a string value, it specifies a tile pattern (see below).
- `border` is a numeric expression in `[0–255]` . It specifies the attribute of the fill boundary (see below).
- `background` is a string expression that represents a background tile pattern to ignore when determining boundaries (see below).

Tile patterns

A tile pattern can be specified by a string of up to 255 characters. The interpretation of the string depends on the number of bits per pixel and on the current screen mode.

1 bit per pixel (e.g. `SCREEN 2`)

Here is an example:

76543210	Byte value
*.......	&h80
.*......	&h40
..*.....	&h20
...*....	&h10
....*...	&h08
.....*..	&h04
......*.	&h02

This diagonal stripe pattern can thus be produced with

```
PAINT (0, 0), CHR$(128)+CHR$(64)+CHR$(32)+CHR$(16)+CHR$(8)+CHR$(4)+CHR$(2)
```

`SCREEN 7 , 8 , 9`

The tile pattern is always 8 pixels wide. The first character in the pattern string contains the first bit of each of these 8 pixels, the second character contains the second bits, etc. For example, in a 2-bits-per-pixel mode, four colour attributes can be used in the pattern. To create a diagonal stripe pattern of the same shape, in attribute `&h03` , we now need a tile string that is twice as long:

Attribute bit	76543210	Byte value
0	*.......	&h80
1	*.......	&h80
0	.*......	&h40
1	.*......	&h40
0	..*.....	&h20
1	..*.....	&h20
0	...*....	&h10
1	...*....	&h10
0	....*...	&h08
1	....*...	&h08
0	.....*..	&h04
1	.....*..	&h04
0	......*.	&h02
1	.......*.	&h02

If the pattern string is truncated before all bits of the last line have been defined, the remaining bits will be zero.

`SCREEN 1 , 3 , 4 , 5 , 6`

Each row of the tile pattern represents a screen row. Colours are encoded in consecutive bits; the more bits per pixel, the narrower the pattern is. For 2 bits per pixel, the pattern is 4 pixels wide; for 4 bits per pixel it is 2 pixels wide. The following pattern string encodes a diagonal dotted stripe in two colours:

3210	76543210	Byte value
2000	*.......	&h80
1000	.*......	&h40
0200	..*.....	&h20
0100	...*....	&h10
0020	....*...	&h08
0010	.....*..	&h04
0002	......*.	&h02

The tile pattern is anchored to the screen; imagine a grid starting at `(0,0)` and covering the screen. Whenever an area is tile-filled, the tiles are put into this grid. In this way, adjacent areas will have continuous tiling even if they were filled from different seed points.

Boundaries

A solid flood fill stops at pixels that have the same attribute as the fill or that have the specified border attribute, if specified. A tiling flood fill stops at the specified border attribute; if no border attribute is specified, it stops at the current foreground attribute. A tiling flood fill also stops at scan line intervals that are the same as the tiling pattern for that line, unless a background pattern is specified and the interval also equals the background pattern for that line.

Errors

- In `SCREEN 7` , `8` , `9` , if *background* equals *attrib* up to the length of *attrib* : `Illegal function call` .
- *background* has a number value: `Illegal function call` .
- *border* , *x* , or *y* have a string value: `Type mismatch` .
- *border* , *x* , or *y* are not in `[-32768—32767]` : `Overflow` .
- *border* is not in `[0—255]` : `Illegal function call` .
- *attrib* is numeric and not in `[-32768—32767]` : `Overflow` .
- *attrib* is numeric and not in `[0—255]` : `Illegal function call` .

PALETTE

`PALETTE` [*attrib*, *colour*]

Assigns a colour to an attribute. All pixels with that attribute will change colour immediately. If no parameters are specified, `PALETTE` resets to the initial setting.

Parameters

- `attrib` is a numeric expression between 0 and the current palette size, less one.
- `colour` is a numeric expression between -1 and the maximum number of colours for the current screen mode, less one. If `colour` equals -1, the palette remains unchanged.

Errors

- `attrib` or `colour` has a string value: `Type mismatch`.
- `attrib` or `colour` is not in `[-32768—32767]` : `Overflow`
- `attrib` or `colour` is not in range: `Illegal function call`

PALETTE USING

```
PALETTE USING int_array_name {(|[} start_index {)|]}
```

Assigns new colours to all attributes.

Parameters

- `int_array_name` is a single- or multidimensional array of integers (`%`) that will supply the new values for the palette.
- `start_index` is a numeric expression that indicates at which index in the array to start mapping to the palette.

Notes

- Array values are assigned to palette entries in the order in which they are stored in memory. See Arrays for details about the layout of arrays in memory.
- If an array entry has value `-1` , the matching attribute is left unchanged.

Errors

- `int_array_name` has not been allocated: `Illegal function call` . The array will not be automatically allocated.
- `int_array_name` is not an integer array: `Type mismatch` .
- `int_array_name` is too short: `Illegal function call` .
- `start_index` has a string value: `Type mismatch` .
- `start_index` is not in `[-32768—32767]` : `Overflow`
- `start_index` is outside array dimensions: `Subscript out of range`

PCOPY

```
PCOPY src, dst
```

Copies the screen page `src` to `dst` . All text and graphics on `dst` is replaced by those of `src` .

Parameters

- `src` and `dst` are numeric expressions between 0 and the current video mode's number of pages, less one.

Errors

- `src` or `dst` has a string value: `Type mismatch` .
- `src` or `dst` is not in `[-32768—32767]` : `Overflow` .
- `src` or `dst` is out of range: `Illegal function call` .

`PEN` (statement)

```
PEN {ON|OFF|STOP}
```

Controls event trapping and read access of the light pen (emulated through the mouse in PC-BASIC). `PEN ON` switches pen reading and trapping on. `PEN OFF` switches it off. `PEN STOP` suspends `PEN` event trapping until `PEN ON` is executed. Up to one event can be triggered during suspension, provided that event handling was switched on prior to suspension. The event triggered during suspension is handled immediately after the next `PEN ON` statement.

`PLAY` (event switch)

```
PLAY {ON|OFF|STOP}
```

- `ON` : enables `ON PLAY` event trapping of the music queue.
- `OFF` : disables trapping.
- `STOP` : halts trapping until `PLAY ON` is used. Events that occur while trapping is halted will trigger immediately when trapping is re-enabled.

PLAY (music statement)

```
PLAY [mml_string_0] [, [mml_string_1] [, mml_string_2]]
```

Plays the tune defined by the Music Macro Language strings `mml_string_0, ...` .

Unless `syntax={tandy | pcjr}` is set, only the single-voice syntax is available. The three separate MML strings correspond to the three voices of the PCjr/Tandy sound adapter. The notes in these strings are played synchronously.

Parameters

- `mml_string_0` , `mml_string_1` , `mml_string_2` are string expressions in MML.
- At least one parameter must be provided and the statement must not end in a comma.

Music Macro Language reference
Notes and Pauses

Command	Effect
`{A|B|C|D|E|F|G}[#|+|-][m]`	Play a note. `+` or `#` indicates sharp. `-` indicates flat. m is a numeric literal and indicates duration of an mth note. m is in the range `[0-64]`. If $m=0$ or omitted, use the default length.
`Nn`	Play note n, in the range `[0-84]` (7 octaves). $n = 0$ means rest.
`On`	Set the current octave to n, in the range `[0-6]`. Default is 4.
`>`	Increase the current octave by 1, with a maximum of 6.
`<`	Decrease the current octave by 1, with a minimum of 0.
`Pn`	Pause for the duration of an nth note. n is in the range `[0-64]`. If $n=0$, this has no effect.

Timing commands

Command	Effect
.	Increase the duration of the preceding note by 1/2 times its normal duration. Periods can be repeated to increase duration further.
L*n*	Set the duration of following note to an *n*th note. (*n*=4 is a quarter note, etc.) *n* is in the range [1–64].
MN	Normal: 7/8 of the duration is sound, with 1/8 silence. Default mode.
ML	Legato: full duration is sound.
MS	Staccato: 3/4 of the duration is sound, with 1/4 silence.
T*n*	Sets the tempo to *n* L4s per minute. *n* is in the range [32–255]. Default is 120.

Background-mode commands

These commands affect `SOUND` , `PLAY` and `BEEP`

Command	Effect
MB	Turns on background mode; sound commands exit without waiting for the music to finish. The music keeps playing while other commands are executed. There can be up to 32 notes in the background music queue; if more notes are played, `PLAY` will block until there are only 32 left. Note that the gaps between notes in the default articulation and in staccato are counted as separate notes on the queue.
MF	Turns off background mode; sound commands block. Default mode.

Subroutine command

Command	Effect
X*s*	Execute substring. *s* is one of the following: • a string variable name followed by a `;` • the result of `VARPTR$()` on a string variable

Volume control

Volume control is available on `syntax={tandy | pcjr}` only:

Command	Effect
V*n*	Set the volume to *n*, in the range [-1–15]. -1 means full volume. If `SOUND ON` has not been executed, this has no effect.

MML Parameters

Numeric variables `n` in the commands above can be:

- an integer literal, e.g. `PLAY "L4G"`
- a numeric variable name or array element `var` preceded by `=` and followed by `;` . For example, `PLAY "L=VAR;G"` or `PLAY "L=A(1);G"`
- the result of `VARPTR$(var)` preceded by `=` . For example, `PLAY "L=" + VARPTR$(VAR) + "G"`

Note that only number *literals* may follow named notes and dereferencing variables or arrays is not allowed there. It is an error to write `PLAY "G=VAR;"` or `PLAY "G=" + VARPTR$(VAR)` . Use `PLAY "G4"` or `PLAY "L=VAR;G"` or `PLAY "L=" + VARPTR$(VAR) + "G"` instead.

Errors

- `mml_string` has a numeric value: `Type mismatch` .
- `mml_string` has errors in the MML: `Illegal function call` .
- A variable in an MML string is of incorrect type: `Type mismatch` .
- No MML string is specified: `Missing operand` .
- If `SOUND ON` has not been executed, using the three-voice syntax will raise `Syntax error` .

POKE

```
POKE address, value
```

Sets the value of the memory byte at `segment * 16 + address` to `value`, where `segment` is the current segment set with `DEF SEG`.

Parameters

- `address` is a numeric expression in `[-32768—65535]`. Negative values are interpreted as their two's complement.
- `value` is a numeric expression in `[0—255]`.

Notes

- The memory is only partly emulated in PC-BASIC. See Memory model for supported addresses. Outside emulated areas of memory, this statement has no effect.

Errors

- `address` or `value` has a string value: `Type mismatch`.
- `address` is not in `[-32768—65535]`: `Overflow`.
- `value` is not in `[-32768—32767]`: `Overflow`.
- `value` is not in `[0—255]`: `Illegal function call`.

`PSET` and `PRESET`

{ **PSET** | **PRESET** } [**STEP**] (*x*, *y*) [, *attrib*]

Change the attribute of a pixel on the screen at position `(x, y)`. If `STEP` is specified, `(x, y)` is an offset from the current position.

If `attrib` is between 0 and the screen mode's palette size, the pixel is changed to attribute `attrib`. If `attrib` is larger than the palette size, the pixel's attribute is changed to the highest legal attribute value. If `attrib` is not specified, `PSET` changes the attribute to the current foreground attribute while `PRESET` changes it to zero.

Parameters

- `x`, `y` are numeric expressions in `[-32768–32767]`.
- `attrib` is a numeric expression in `[0–255]`.

Errors

- `x` or `y` has a string value: `Type mismatch`.
- `attrib`, `x` or `y` or the physical coordinates they translate into are not in `[-32768–32767]`: `Overflow`.
- `attrib` is not in `[0–255]`: `Illegal function call`.

`PRINT` **and** `LPRINT`

```
{LPRINT|{PRINT|?} [# file_num,]} [expr_0|;|,|SPC(n)|TAB(n)] ... [USING format; uexpr_0
    [{;|,} uexpr_1] ... [;|,]]
```

Writes expressions to the screen, printer, or file. If `LPRINT` is used, output goes to `LPT1:` . If `file_num` is specified, output goes to the file open under that number. `?` is a shorthand for `PRINT` .

When writing a string expression to the screen, the following control characters have special meaning. Other characters are shown as their corresponding glyph in the current <u>codepage</u>.

Code point	Control character	Effect
&h07	BEL	Beep the speaker.
&h08	BS	Erase the character in the previous column and move the cursor back.
&h09	HT	Jump to the next 8-cell tab stop.
&h0A	LF	Go to the leftmost column in the next row; connect the rows to one logical line.
&h0B	VT	Move the cursor to the top left of the screen.
&h0C	FF	Clear the screen.
&h0D	CR	Go to the leftmost column in the next row.
&h1C	FS	Move the cursor one column to the right.
&h1D	GS	Move the cursor one column to the left.
&h1E	RS	Move the cursor one row up.
&h1F	US	Move the cursor one row down.

Expressions can optionally be separated by one or more of the following keywords:

Keyword	Effect
`;`	Attaches two expressions tight together; strings will be printed without any space in between, numbers will have one space separating them, in addition to the space or minus sign that indicate the sign of the number.
`,`	The expression after will be positioned at the next available zone. The output file is divided in 14-character zones; if the width of the file is not a multiple of 14, the remaining spaces are unused and the first zone of the next line is used instead. If the file has a width of less than 14 characters, the zones are determined as if the file were wrapping continuously.
`SPC(n)`	Produces n spaces, where n is a numeric expression. if n is less than zero, it defaults to zero. If n is greater than the file width, it is taken modulo the file width.
`TAB(n)`	Moves to column n, where n is a numeric expression. if n is less than zero, it defaults to zero. If n is greater than the file width, it is taken modulo the file width. If the current column is greater than n, TAB moves to column n on the next line.

If the print statement does not end in one of these four separation tokens, a newline is printed after the last expression. String expressions can be separated by one or more spaces, which has the same effect as separating by semicolons.

Format string syntax

A `USING` declaration occurs at the end of an `[L]PRINT[#]` statement and writes a formatted string to the screen, printer or file. The following tables list the format tokens that can be used inside the `format` string.

`_`	Escape character; causes the next character in the format string to be printed as is rather than interpreted as a format token.

For string expressions:

`!`	Prints the first character of a string.
`\\`	Prints 2 or more characters of a string. A greater number of characters is selected by separating the \s by spaces.
`&`	Prints the whole string.

For numeric expressions, the format string specifies a width and alignment.

#	Indicate a position for a digit.
.	Indicate the decimal point.
,	Before the decimal point: cause digits to be grouped in threes separated by commas. After the decimal point it is not a token. Provides one digit position.

The number of characters in the field must not exceed 24.

Tokens preceding the number field:

+	Cause the sign to be printed for positive as well as negative numbers. The sign is to be printed to the left of the number.
**	Cause any leading spaces to be replaced with *s. Provides two digit positions.
$$	Cause a $ to be printed to the left of the number. Provides one digit position.

Tokens trailing the number field:

+	Cause the sign to be printed for positive as well as negative numbers. The sign will be printed to the right of the number.
–	Cause the sign for negative numbers to be printed to the right of the number. Note that – preceding the field is not a token but printed literally.
^^^^	Specify that scientific notation E+00 is to be used.

Numeric expressions are always fully printed, even if they do not fit in the positions specified. If the number does not fit in the allowed space, a % is printed preceding it.

- If there are more expressions than format fields, the format string is wrapped around.
- Expressions may be separated with semicolons or commas; the effect is the same.
- If the USING declaration ends in a comma or semicolon, no newline is printed at the end.
- After a USING declaration, other elements of the PRINT syntax such as SPC(and TAB(can not be used.

Parameters
- *expr_0*, *expr_1*, ... are expressions of any type.
- *format* is a string expression that specifies the output format.
- *uexpr_0*, *uexpr_1*, ... are expressions matching a token in the format string.

Notes
- If an error is raised, the output before the error was encountered is printed as

normal.

- In GW-BASIC, when formatting a number with a dollar sign, if the number is in the range `[-10000–-32767]` and does not fit in the width of the number field, the minus sign is omitted. This is not implemented in PC-BASIC.

Errors

- *n* has a string value: `Type mismatch`.
- *n* is not in `[-32768–65535]`: `Overflow`.
- The format string contains no tokens: `Illegal function call`.
- An expression doesn't match the corresponding format token type: `Type mismatch`.
- A number field in the format string exceeds 24 characters: `Illegal function call`.
- A number field in the format string contains no `#` characters: `Illegal function call`.

PUT (files)

```
PUT [#] file_number [, record_number]
```

Writes a record to the random-access file `file_number` at position `record_number`.

Parameters

- `file_number` is a numeric expression that yields the number of an open random-access file. The `#` is optional and has no effect.
- `record_number` is a numeric expression in `[1–33554432]` (`2^25`) and is interpreted as the record number.

Notes

- The record number is stored as single-precision; this precision is not high enough to distinguish single records near the maximum value of `2^25`.

Errors

- `record_number` is not in `[1–33554432]`: `Bad record number`.
- `file_number` is not in `[0–255]`: `Illegal function call`.
- `file_number` is not the number of an open file: `Bad file mode`.
- `file_number` is open under a mode other than `RANDOM`: `Bad file mode`.
- `file_number` is not specified: `Missing operand`.

PUT (communications)

```
PUT [#] com_file_number [, number_bytes]
```

Writes `number_bytes` bytes to the communications buffer opened under file number `com_file_number`. `number_bytes` is a numeric expression between 1 and the `COM` buffer length, inclusive.

Notes

- In GW-BASIC, `Device I/O error` is raised for overrun error, framing error, and break interrupt. `Device fault` is raised if DSR is lost during I/O. A `Parity error` is raised if parity is enabled and incorrect parity is encountered. This is according to the manual; it is untested.

Errors

- `bytes` is less than 1: `Bad record number`.
- `bytes` is less than `32768` and greater than the `COM` buffer length: `Illegal function call`.
- `com_file_number` is not specified: `Missing operand`.
- `com_file_number` is not in `[0–255]`: `Illegal function call`.
- `com_file_number` is not the number of an open file: `Bad file number`.
- The serial input buffer is full, i.e. `LOF(com_file_number) = 0` and `LOC(com_file_number)=255`: `Communication buffer overflow`.

`PUT` (graphics)

```
PUT (x0, y0), array_name [, {PSET|PRESET|AND|OR|XOR}]
```

Displays an array to a rectangular area of the graphics screen. Usually, `PUT` is used with arrays that have been stored using `GET`. See `GET` for the format of the array.

The keywords have the following effect:

`PSET`	Overwrite the screen location with the new image
`PRESET`	Overwrite the screen location with the inverse image
`AND`	Combines the old and new attributes with bitwise AND
`OR`	Combines the old and new attributes with bitwise OR
`XOR`	Combines the old and new attributes with bitwise XOR

Parameters
- `array_name` is a numeric array.
- `x0`, `y0` are numeric expressions.

Errors
- The array does not exist: `Illegal function call`.
- `array_name` refers to a string array: `Type mismatch`.
- `x0`, `y0` are string expressions: `Type mismatch`.
- `x0`, `y0` are not in `[-32768–32767]`: `Overflow`.
- `x0`, `y0` is outside the current `VIEW` or `WINDOW`: `Illegal function call`

RANDOMIZE

`RANDOMIZE` [*expr*]

Seeds the random number generator with `expr` . If no seed is specified, `RANDOMIZE` will prompt the user to enter a random seed. The user-provided value is rounded to an integer. The random seed is formed of the last two bytes of that integer or `expr` . If `expr` is a float (4 or 8 bytes), these are `XOR` ed with the preceding 2. The first 4 bytes of a double are ignored. The same random seed will lead to the same sequence of pseudorandom numbers being generated by the `RND` function.

Parameters

- `expr` is a numeric expression.

Notes

- For the same seed, PC-BASIC produces the same pseudorandom numbers as GW-BASIC 3.23.
- The random number generator is very poor and should not be used for serious purposes. See `RND` for details.

Errors

- `expr` has a string value: `Illegal function call` .
- The user provides a seed outside `[-32768–32767]` at the prompt: `Overflow` .

READ

`READ` *var_0* [, *var_1*] ...

Assigns data from a `DATA` statement to variables. Reading starts at the current `DATA` position, which is the `DATA` entry immediately after the last one read by previous `READ` statements. The `DATA` position is reset to the start by the `RUN` and `RESTORE` statements.

Parameters

- `var_0` , `var_1` are variables or array elements.

Errors

- Not enough data is present in `DATA` statements: `Out of DATA` .
- The type of the variable is not compatible with that of the data entry being read: a `Syntax error` occurs on the `DATA` line.

REM

```
{REM|'} [anything]
```

Ignores everything until the end of the line. The `REM` statement is intended for comments. Everything after `REM` will be stored in the program unaltered and uninterpreted. `'` (apostrophe) is an alias for `:REM'`; it can be placed at any point in the program line and will ensure that the rest of the line is ignored.

Note that a colon `:` does not terminate the `REM` statement; the colon and everything after it will be treated as part of the comment.

RENUM

`RENUM` [*new*|.] [, [*old*|.] [, *increment*]]

Replaces the line numbers in the program by a systematic enumeration starting from `new` and increasing by `increment`. If `old` is specified, line numbers less than `old` remain unchanged. `new`, `old` are line numbers; the dot `.` signifies the last line edited. `increment` is a line number but must not be a dot or zero.

Notes

- Line numbers afer the following keywords will be renumbered: `AUTO`, `EDIT`, `ELSE`, `ERL`, `DELETE`, `GOSUB`, `GOTO`, `LIST`, `LLIST`, `RENUM`, `RESTORE`, `RESUME`, `RETURN`, `RUN`, `THEN`.
- Any line numbers in `CHAIN` statements will not be renumbered; note that these line numbers refer to another program.
- All arguments of `RENUM` or `AUTO` statements in a program will be renumbered, including any line number offsets or increments, even though that does not make much sense.
- A zero line number following the keywords `ERROR GOTO` will not be renumbered.
- If a referenced line number does not exist in the program, a message `Undefined line` *ref* `in` *old_line* is printed. Here, `old_line` is the line number prior to renumbering. The referenced line number will be left unchanged, but the line's old line number will be renumbered.

Errors

- Any of the parameters is not in `[0—65529]`: `Syntax error`.
- Any of the newly generated line numbers is greater than `65529`: `Illegal function call`. The line numbers up to the error have not been changed.
- `increment` is empty or zero: `Illegal function call`.
- `old` is specified and `new` is less than or equal to an existing line number less than old: `Illegal function call`.

RESET

Closes all open files.

Notes

- Official GW-BASIC documentation and many other sources state that `RESET` closes all files *on disk devices*. However, in reality GW-BASIC 3.23 also closes files on tape and any other device, making this statement identical to `CLOSE` with no arguments. PC-BASIC follows this behaviour.

RESTORE

`RESTORE` [*line*]

Resets the `DATA` pointer. *line* is a line number. If *line* is not specified, the `DATA` pointer is reset to the first `DATA` entry in the program. If it is specified, the `DATA` pointer is reset to the first `DATA` entry in or after *line*.

Errors

- *line* is not an existing line number: `Undefined line number`.

RESUME

`RESUME` [**0**|**NEXT**|*line*]

Continues normal execution after an error handling routine. If `0` or no option is specified, re-executes the statement that caused the error. If `NEXT` is specified, executes the statement following the one that caused the error. If *line* is specified, it must be a valid line number.

Errors

- `RESUME` is encountered outside of an error trapping routine: `RESUME without error`.
- The program ends inside an error trapping routine without a `RESUME` or `END` statement: `No RESUME`.
- *line* is not an existing line number: `Undefined line number`.

RETURN

`RETURN` [*line*]

Returns from a `GOSUB` subroutine. If *line* is not specified, `RETURN` jumps back to the statement after the `GOSUB` that jumped into the subroutine. If *line* is specified, it must be a valid line number. `RETURN` jumps to that line (and pops the `GOSUB` stack). When returning from an error trapping routine, `RETURN` re-enables the event trapping which was stopped on entering the trap routine.

Errors

- *line* is not an existing line number: `Undefined line number`.

RMDIR

RMDIR *dir_spec*

Removes an empty directory on a disk device.

Parameters

- The string expression *dir_spec* is a valid <u>file specification</u> that specifies the path and name of the directory.

Errors

- *dir_spec* has a numeric value: `Type mismatch`.
- *dir_spec* is an empty string: `Bad file name`.
- No matching path is found: `Path not found`.
- Directory to remove is not empty: `Path/File access error`.

RSET

```
RSET string_name = expression
```

Copies a string value into an existing string variable or array element. The value will be right-justified and any remaining characters are replaced by spaces.

Parameters

- `string_name` is a string variable or array element.
- `expression` is a string expression.

Notes

- If `expression` has a value that is longer than the length of the target variable, it is truncated at the tail to the length of the target variable.
- If `string_name` has not been allocated before, this statement has no effect.
- Use `LSET` , `RSET` or `MID$` to copy values into a `FIELD` buffer.
- If `LET` is used on a `FIELD` variable instead of `L|RSET` , the variable is detached from the field and a new, normal string variable is allocated.

Errors

- `string_name` is not a string variable: `Type mismatch` .
- `expression` does not have a string value: `Type mismatch` .

RUN

```
RUN [line_number [anything]|file_spec [, R]]
```

Executes a program. Existing variables will be cleared and any program in memory will be erased. `RUN` implies a `CLEAR` If `,R` is specified after `file_spec`, files are kept open; if not, all files are closed.

Parameters

- `line_number` is a valid line number in the current program. If specified, execution starts from this line number. The rest of the `RUN` statement is ignored in this case.
- The string expression `file_spec`, if specified, is a valid file specification indicating the file to read the program from.

Errors

- `line_number` is not a line number in the current program: `Undefined line number`.
- `file_spec` cannot be found: `File not found`.
- `file_spec` is an empty string: `Bad file number`.
- A loaded text file contains lines without line numbers: `Direct statement in file`.

```
SAVE file_spec [, {A|P}]
```

Stores the current program in a file.

- If `,A` is specified, the program will be saved in plain text format. In this case, program execution will stop and control will be returned to the user.
- If `,P` is specified, the program will be saved in protected format. When a protected program is loaded in GW-BASIC, it cannot be `LIST` ed or `SAVE` d in non-protected format.
- If neither is specified, the program will be saved in tokenised format.

Parameters

- The string expression `file_spec` is a valid file specification indicating the file to store to.

Errors

- `file_spec` has a number value: `Type mismatch` .
- `file_spec` is an empty string: `Bad file number` .
- `file_spec` contains disallowed characters: `Bad file number` (on `CAS1:`); `Bad file name` (on disk devices).
- `hide-protected` is enabled, the current program is protected and `,P` is not specified: `Illegal function call` .

SCREEN (statement)

`SCREEN [mode] [, [colorburst] [, [apage] [, [vpage] [, erase]]]]`

Change the video mode, composite colorburst, active page and visible page. Video modes are described in the Video Modes section.

Parameters

- `mode` is a numeric expression that sets the screen mode.
- `colorburst` is a numeric expression. See notes below.
- `apage` is a numeric expression that sets the active page.
- `vpage` is a numeric expression that sets the visible page.
- `erase` is a numeric expression in the range `[0, 1, 2]`. It is only legal with `syntax={pcjr, tandy}`. See notes below.

Video modes

The video modes are as follows:

`SCREEN 0` **Text mode**

    80x25 or 40x25 characters of 8x16 pixels
    16 attributes picked from 64 colours
    Attributes 16-31 are blinking versions of 0-15
    4 pages `ega`

`SCREEN 1` **CGA colour**

    320x200 pixels
    40x25 characters of 8x8 pixels
    4 attributes picked from 16 colours; 2 bits per pixel
    1 page `ega` 2 pages `pcjr` `tandy`

`SCREEN 2` **CGA monochrome**

    640x200 pixels
    80x25 characters of 8x8 pixels
    2 attributes picked from 16 colours; 1 bit per pixel
    1 page `ega` 2 pages `pcjr` `tandy`

`SCREEN 3` **Low-res 16-colour** `pcjr` `tandy`

    160x200 pixels
    20x25 characters of 8x8 pixels
    16 attributes picked from 16 colours; 4 bits per pixel
    2 pages

`SCREEN 3` **Hercules monochrome** `hercules`

    720x348 pixels

    80x25 characters of 9x14 pixels (with bottom line truncated by 2 px)

    2 attributes; 1 bit per pixel

    2 pages

`SCREEN 3–255` **Altissima risoluzione** `olivetti`

    640x400 pixels

    80x25 characters of 8x16 pixels

    2 attributes of which one picked from 16 colours; 2 bits per pixel

    1 page

`SCREEN 4` **Med-res 4-colour** `pcjr` `tandy`

    320x200 pixels

    40x25 characters of 8x8 pixels

    4 attributes picked from 16 colours; 2 bits per pixel

    2 pages

`SCREEN 5` **Med-res 16-colour** `pcjr` `tandy`

    320x200 pixels

    40x25 characters of 8x8 pixels

    16 attributes picked from 16 colours; 4 bits per pixel

    1 page

    Note: a minimum of 32768 bytes of video memory must be reserved to use this video mode. Use the statement `CLEAR ,,,32768!` or the option `video-memory=32768` .

`SCREEN 6` **High-res 4-colour** `pcjr` `tandy`

    640x200 pixels

    80x25 characters of 8x8 pixels

    4 attributes picked from 16 colours; 2 bits per pixel

    1 page

    Note: a minimum of 32768 bytes of video memory must be reserved to use this video mode. Use the statement `CLEAR ,,,32768!` or the option `video-memory=32768` .

`SCREEN 7` **EGA colour** `ega`

    320x200 pixels

    40x25 characters of 8x8 pixels

    16 attributes picked from 16 colours; 4 bits per pixel

    8 pages

`SCREEN 8` **EGA colour** `ega`

    640x200 pixels

    80x25 characters of 8x8 pixels

16 attributes picked from 16 colours; 4 bits per pixel

4 pages

`SCREEN 9` **EGA colour** `ega`

640x350 pixels

80x25 characters of 8x14 pixels

16 attributes picked from 64 colours; 4 bits per pixel

2 pages

`SCREEN 10` **EGA monochrome** `ega` `monitor=mono`

640x350 pixels

80x25 characters of 8x14 pixels

4 attributes picked from 9 pseudocolours; 2 bits per pixel

2 pages

NTSC Composite Colorburst

On CGA, Tandy and PCjr, `colorburst` has the following effects, depending on the type of monitor - RGB (default) or composite:

mode	*colorburst*	CGA mode	Effect (composite)	Effect (RGB)
0	0	0, 2	greyscale	default palette
0	1	1, 3	colour	default palette
1	0	4	colour	default palette
1	1	5	greyscale	alternate palette

On `SCREEN 2`, `colorburst` has no effect; on a composite monitor, colour artifacts can be enabled on this screen through `OUT` (see there). On `SCREEN 3` and up, `colorburst` has no effect.

Erase

By default, if the `mode` changes or the `colorburst` changes between zero and non-zero, the old page and the new page of the screen are cleared. On `syntax={pcjr, tandy}`, the `erase` parameter can be used to change this behaviour. Its values are as follows:

erase	Effect
0	Do not erase any screen page
1 (default)	If the *mode* changes or the *colorburst* changes between zero and non-zero, the old page and the new page of the screen are cleared.
2	If the *mode* changes or the *colorburst* changes between zero and non-zero, all pages of the screen are cleared.

Notes

- At least one parameter must be specified.
- Composite colour artifacts are emulated only crudely in PC-BASIC, and not at all in `SCREEN 1` .

Errors

- No parameters are specified: `Missing operand` .
- Any parameter has a string value: `Type mismatch` .
- Any parameter is not in `[-32768—32767]` : `Overflow` .
- *mode* is not an available video mode number for your video card setting: `Illegal function call` .
- *vpage* , *apage* are not between 0 and the number of pages for the chosen video mode, less one: `Illegal function call` .
- *colorburst* is not in `[0—255]` : `Illegal function call` .
- *erase* is not in `[0, 1, 2]` : `Illegal function call` .

SHELL

SHELL [*command*]

Starts an operating system subshell on the console. If *command* is specified, the command is executed on the shell and execution returns to the program.

To enable this statement, the `shell` option must be set to a valid command interpreter.

Parameters

- *command* is a string expression.

Notes

- Be careful when enabling this command, as it allows the running BASIC program full access to your files and operating system.

Errors

- `shell` option is not specified: `Illegal function call` .
- *command* has a number value: `Type mismatch` .
- All output from the operating system subshell, including error messages, is displayed on the PC-BASIC screen.

SOUND (tone)

```
SOUND frequency, duration [, volume [, voice]]
```

Produces a sound at `frequency` Hz for `duration`/18.2 seconds. On PCjr and Tandy, the volume and voice channel can additionally be specified.

If `PLAY "MB"` has been executed, `SOUND` plays in the background. If `PLAY "MF"` has been executed, sound plays in the foreground and the interpreter blocks until the sound is finished. Foreground mode is default. Unlike `PLAY`, the sound played by the most recent `SOUND` statement always plays in the background, even if `PLAY "MF"` has been entered. In background mode, each `SOUND` statement counts as 1 toward the length of the queue reported by the `PLAY` function.

Parameters
- `frequency` is a numeric expression in `[37–32767]` or `0` (for `syntax={advanced | pcjr}`) or in `[-32768–32767]` (for `syntax=tandy`).
- `duration` is a numeric expression in `[0–65535]`.
- `volume` is a numeric expression in `[-1–15]`. 0 is silent, 15 is full volume; every step less reduces the volume by 2 dB. -1 is also full volume. (For `syntax={pcjr | tandy}`).
- `voice` is a numeric expression in `[0–2]`, indicating which of the three tone voice channels is used for this sound. (For `syntax={pcjr | tandy}`)

Notes
- On PCjr and Tandy, Frequencies below 110 Hz are played as 110 Hz.
- If `duration` is zero, any active background sound is stopped and the sound queue is emptied.
- If `duration` is zero, `volume` and `voice` must not be specified.
- If `duration` is less than `.022` but nonzero, the sound will be played in background and continue indefinitely until another sound statement is executed. This is also the behaviour for negative `duration`.
- If `frequency` equals `32767` or `0`, a silence of length `duration` is queued.

Errors
- Any argument has a string value: `Type mismatch`.
- `frequency` is not in its allowed range, and `duration` is not zero: `Illegal function call`.
- `duration` is zero and more than two arguments are specified: `Syntax error`.
- `syntax={ pcjr | tandy }` is not set and more than two arguments are specified:

`Syntax error` .

- *frequency* is not in `[-32768–32767]` : `Overflow` .
- *duration* is not in `[-65535–65535]` : `Illegal function call` .
- *volume* is not in `[0–15]` : `Illegal function call` .
- *voice* is not in `[0–2]` : `Illegal function call` .

SOUND (switch)

`SOUND {ON|OFF}`

Switches the external speaker on or off and toggles the availability of advanced sound capabilities on PCjr and Tandy. This includes 3-voice sound, noise generation and volume control. Clears the background music queue.

Notes

- Only available with `syntax={pcjr | tandy}` .
- On PC-BASIC, both the internal and the external speaker are emulated through the same sound system.

Errors

- This statement is used and `syntax={ pcjr | tandy }` is not set: `Syntax error` .

STOP

`STOP`

Breaks program execution, prints a `Break` message on the console and returns control to the user. Files are not closed. It is possible to resume program execution at the next statement using `CONT` .

STRIG (switch)

`STRIG {ON|OFF}`

Has no effect.

`STRIG` **(event switch)**

`STRIG`[ ]**(***button***)** {**ON**|**OFF**|**STOP**}

Switches event trapping of the joystick trigger `button` `ON` or `OFF` . `STRIG (button) STOP` suspends event trapping until `STRIG (button) ON` is executed. Up to one event can be triggered during suspension, provided that event handling was switched on prior to suspension. The event triggered during suspension is handled immediately after the next `STRIG (button) ON` statement.

`button`	return value
0	1st joystick 1st trigger
2	2nd joystick 1st trigger
4	1st joystick 2nd trigger
6	2nd joystick 2nd trigger

Parameters

- `button` is a numeric expression in `[0, 2, 4, 6]` .

Errors

- `button` has a string value: `Type mismatch` .
- `button` is not in `[-32768–32767]` : `Overflow` .
- `button` is not in `[0, 2, 4, 6]` : `Illegal function call` .

SWAP

```
SWAP var_0, var_1
```

Exchanges variables `var_0` and `var_1` .

Notes

- The variables are exchanged by reference. If, for example, `var_0` is a `FIELD` variable and `var_1` is not, then `SWAP` will reverse those roles.

Parameters

- `var_0` and `var_1` are variables or array elements of the same type. `var_1` must have been previously defined.

Errors

- `var_1` is undefined: `Illegal function call` . Note that no error is raised if `var_0` is undefined, and that after this error both variables will be defined.
- The types of `var_0` and `var_1` are not the same: `Type mismatch` .

SYSTEM

```
SYSTEM
```

Exits the interpreter.

Notes

- `SYSTEM` quits the PC-BASIC interpreter immediately without further interaction. Any unsaved program or data will be lost.

TERM

```
TERM
```

Load and run the program defined by the `term` option. By default, as on the IBM PCjr, this is a built-in serial terminal emulator application. This statement is only available with `syntax={pcjr|tandy}` .

Errors

- If `term` is not set, this statement raises `Internal error` .
- If `syntax` is not set to `pcjr` or `tandy` , this keyword is not present. Calling `TERM` will raise `Syntax error` .

`TIME$` (statement)

```
TIME$ = time
```

Sets the current BASIC time to `time`.

Parameters

- Time is a string expression of the form `"HH{:|.}mm{:|.}ss"` where `0 <= HH < 24`, `0 <= mm < 60` and `0 <= ss < 60`. Each position may have one or two characters.

Notes

- PC-BASIC stores an offset to the system time and uses this for future calls to `TIME$` and `DATE$` functions in the same interpreter session. The system time is not changed, unlike GW-BASIC under MS-DOS.

Errors

- `time` has a numeric value: `Type mismatch`.
- `time` is not of the correct form: `Illegal function call`.

`TIMER` (statement)

```
TIMER {ON|OFF|STOP}
```

- `ON`: enables `ON TIMER` event trapping of the timer clock.
- `OFF`: disables trapping.
- `STOP`: halts trapping until `TIMER ON` is used. Events that occur while trapping is halted will trigger immediately when trapping is re-enabled.

`TRON` and `TROFF`

```
{TRON|TROFF}
```

Turns line number tracing on or off. If line number tracing is on, BASIC prints a tag `[100]` to the console when program line `100` is executed, and so forth.

Notes

- Tracing is turned off by the `NEW` and `LOAD` statements.

`UNLOCK` [`#`] *file_number* [`,` *record_0*]

`UNLOCK` [`#`] *file_number*`,` [*record_0*] `TO` *record_1*

Unlocks a file or part of it that has previously been locked with `LOCK` .

Parameters

- *file_number* is a numeric expression in `[0—255]` .
- *record_0* and *record_1* are numeric expressions in `[1—2^25-2]` .

Errors

- Any parameter has a string value: `Type mismatch` .
- *file_number* is not in `[-32768—32767]` : `Overflow` .
- *file_number* is not in `[0—255]` : `Illegal function call` .
- *file_number* is not an open file: `Bad file number` .
- If *file_number* is open for `RANDOM` , `LOCK` and `UNLOCK` statements must match in terms of *record_0* and *record_1* . An non-matching `UNLOCK` will raise `Permission denied` .
- *record_0* or *record_1* is not in `[1—2^25-2]` : `Bad record number` .

VEW

```
VIEW [[SCREEN] (x0, y0)-(x1, y1) [, [fill] [, border]]]
```

Defines a graphics viewport. Graphics drawn outside the viewport will not be shown. `(x0, y0)`, `(x1, y1)` are absolute screen coordinates of two opposing corners of the area.

Unless `SCREEN` is specified, after a `VIEW` statement the coordinate system is shifted such that `(0, 0)` becomes the top left coordinate of the viewport. If `VIEW` is called without arguments, the viewport is reset to the whole screen.

Parameters
- `fill` is an attribute. The viewport will be filled with this attribute.
- `border` is an attribute. A border will be drawn just outside the viewport with this attribute.

Errors
- Any of the parameters has a string value: `Type mismatch`.
- Any of the coordinates is not in `[-32768–32767]`: `Overflow`.
- Any of the coordinate pairs is outside the physical screen: `Illegal function call`.

VIEW PRINT

```
VIEW PRINT top_row TO bottom_row
```

Defines the text scrolling area of the screen. `LOCATE` statements, cursor movement and scrolling will be limited to the scrolling area.

Parameters
- `top_row` and `bottom_row` are numeric expressions in `[1–24]`.

Notes
- If `syntax={pcjr | tandy}` and `KEY OFF` is set, `bottom_row` may be 25. Otherwise, screen row 25 cannot be part of the scrolling area.

Errors
- `top_row` or `bottom_row` is not in `[1–24]`: `Illegal function call`.

WAIT

`WAIT` *port*, *and_mask* [, *xor_mask*]

Waits for the value of `(INP(port) XOR xor_mask) AND and_mask` to become nonzero. Event handling is suspended until `WAIT` returns. If `xor_mask` is not specified, it defaults to `0`.

Notes

- A limited number of machine ports are emulated in PC-BASIC. See `INP`.

Errors

- Any parameter has a string value: `Type mismatch`.
- `port` is not in `[-32768–65535]`: `Overflow`.
- `and_mask` or `xor_mask` are not in `[0–255]`: `Type mismatch`.

WEND

`WEND`

Iterates a `WHILE–WEND` loop: jumps to the matching `WHILE` statement, where its condition can be checked.

Notes

- `WHILE–WEND` loops can be nested. `WEND` jumps to the most recent `WHILE` statement that has not been closed by another `WEND`.

Errors

- All previous `WHILE` statements have been closed by another `WEND` or no `WHILE` statement has been executed before: `WEND without WHILE`.

WHILE

`WHILE` *expr*

Initiates a `WHILE–WEND` loop. If `expr` evaluates to zero, `WHILE` jumps to the statement immediately after the matching `WEND`. If not, execution continues.

Parameters

- `expr` is a numeric expression.

Errors

- No matching `WEND` is found: `WHILE without WEND`.
- `expr` has a string value: `Type mismatch`.

`WIDTH` (console)

```
WIDTH num_columns [, [num_rows] [,]]
```

Sets the screen width to 20, 40 or 80 columns.

Notes

- When changing screen width in graphics mode, the video mode is changed. The following changes occur:

  `SCREEN 1` (40) ↔ `SCREEN 2` (80)

  `SCREEN 7` (40) ↔ `SCREEN 8` (80)

  `SCREEN 7` (40) ← `SCREEN 9` (80)

- Screen width value 20 is only allowed on Tandy and PCjr. Changing to this width changes to `SCREEN 3`. Additionally, the following changes occur:

  `SCREEN 3` (20) → `SCREEN 1` (40)

  `SCREEN 3` (20) → `SCREEN 2` (80)

  `SCREEN 4` (40) → `SCREEN 2` (80)

  `SCREEN 5` (40) ↔ `SCREEN 6` (80)

Parameters

- *num_columns* is either a literal `20`, `40` or `80` or a numeric expression in parentheses. The trailing comma is optional and has no effect.
- *num_rows* is optional and must equal 25. If `syntax={pcjr | tandy}` is set, *num_rows* may be in `[0–25]` but its value is ignored.

Errors

- *num_columns* is a string expression: `Type mismatch`.
- *num_columns* is not in `[-32768–32767]`: `Overflow`.
- *num_columns* is not in `[0–255]`: `Illegal function call`.
- *num_columns* is not a literal and not bracketed: `Illegal function call`.
- *num_rows* is not in its accepted range: `Illegal function call`.

`WIDTH` (devices and files)

```
WIDTH {#file_num,|device_name,|LPRINT} num_columns
```

Sets the line width for a file or a device. When a write operation passes beyond the column width, a `CR LF` sequence is inserted.

If a device is specified, it does not need to have a file open to it; the width setting will be the default width next time a file is opened to that device.

If `device_name` is `"LPT1:"` or `LPRINT` is specified, the device width setting affects `LPRINT` and `LLIST` .

If `device_name` is `"SCRN:"` , `"KYBD:"` , or omitted, the screen width is changed. In this case, `num_columns` must be one of 20, 40 or 80. See the notes at `WIDTH` (console) for side effects.

Parameters

- `file_num` is a numeric expression which is the number of an open file.
- `device_name` is a string expression that is one of `"KYBD:"` , `"SCRN:"` , `"LPT1:"` , `"LPT2:"` , `"LPT3:"` , `"COM1:"` , `"COM2:"` , `"CAS1:"`
- `num_columns` is a numeric expression.

Errors

- `device_name` is not one of the allowed devices: `Bad file name` .
- `device_name` is `"SCRN:"` , `"KYBD:"` and `num_columns` is not 20, 40 or 80: `Illegal function call` .
- `file_num` or `num_columns` are strings: `Type mismatch` .
- `file_num` or `num_columns` are not in `[-32768—32767]` : `Overflow` .
- `file_num` or `num_columns` are not in `[0—255]` : `Illegal function call` .
- `file_num` is not an open file: `Bad file mode` .

WINDOW

`WINDOW` `[[`**`SCREEN`**`]` `(`*`x0`*`,` *`y0`*`)-(`*`x1`*`,` *`y1`*`)]`

Define logical coordinates for the viewport. If `SCREEN` is not specified, the bottom left of the screen is mapped to the lower coordinates; the top right of the screen is mapped to the higher coordinates. If `SCREEN` is specified, the top left of the screen is mapped to the lower coordinates; the bottom right of the screen is mapped to the higher coordinates.

If `WINDOW` is called without arguments, the logical coordinates are reset to the viewport coordinates.

Parameters

- `x0` , `y0` , `x1` , `y1` are numeric expressions.

Errors

- Any of the coordinates have a string value: `Type mismatch` .
- `x0 = x1` or `y0 = y1` : `Illegal function call` .

WRITE

`WRITE` `[`**`#`** *`file_num`*`,]` `[`*`expr_0`* `[{,|;}` *`expr_1`*`]` `...` `]`

Writes values to a file or the screen in machine-readable form. Values are separated by commas and the line is ended with a `CR LF` sequence. Strings are delimited by double quotes `"` . No padding spaces are inserted.

When writing to the screen, the same control characters are recognised as for the `PRINT` statement.

Parameters

- `expr_0, expr_1, ...` are expressions whose value is to be printed.

Errors

- `file_num` has a string value: `Type mismatch` .
- `file_num` is open for `INPUT` : `Bad file mode` .

6.8. Errors and Messages

Errors

1	**NEXT without FOR**
	A NEXT statement has been encountered for which no matching FOR can be found.

2	**Syntax error**
	The BASIC syntax is incorrect. A statement or expression has been mistyped or called in one of many incorrect ways. This error is also raised on a DATA line if a READ statement encounters a data entry of an incorrect format.

3	**RETURN without GOSUB**
	A RETURN statement has been encountered for which no GOSUB call has been made.

4	**Out of DATA**
	A READ statement is attempting to read more data entries than can be found from the current DATA location onward.

5	**Illegal function call**
	A statement, function or operator has been called with parameters outside the accepted range. This error is also raised for a large variety of other conditions – check the reference for the statement or function called.

6	**Overflow**
	A numeric expression result or intermediate value is too large for the required number format.

7	**Out of memory**
	There is not enough free BASIC memory to complete the operation. Too much memory is consumed by the program; variables, arrays and strings, or execution stacks for loops, subroutines or user-defined functions.

8	**Undefined line number**
	A reference is made to a line number that does not exist in the program.

9	**Subscript out of range**
	An array index (subscript) is used that is outside the range reserved for that array by the DIM statement.

10	**Duplicate Definition**

A `DIM` statement is used on an array that has been dimensioned before (either implicitly or explicitly) or `OPTION BASE` is called in a way that conflicts with an earlier implicit or explicit definition of the starting index.

11	**Division by zero**

An attempt is made to divide a number by zero or by a number that is too small to distinguish from zero within the number format's precision.

12	**Illegal direct**

A `DEF FN` statement is being used in direct mode.

13	**Type mismatch**

The expression used is of a type that cannot be converted to the required type for the function or statement. Most commonly, this is raised if a string argument is supplied to a statement or function that expects a number, or vice versa.

14	**Out of string space**

There is not enough free BASIC memory to store the string variable.

15	**String too long**

A string expression result or intermediate value is longer than 255 characters.

16	**String formula too complex**

17	**Can't continue**

The `CONT` statement is used in circumstances where continuing program execution is not possible.

18	**Undefined user function**

The `FN` function is called with a function name for which no definition was made by a `DEF FN` statement.

19	**No RESUME**

The program terminates inside an error trapping routine that has not been closed with `RESUME` or `END`.

20	**RESUME without error**

A `RESUME` statement is encountered while the program is not executing an error trapping routine.

21	*unused*

22   Missing operand

An operator expression misses an operand or a function or statement is not supplied with sufficient parameters.

23   Line buffer overflow

An INPUT or LINE INPUT statement encountered an input string longer than 255 characters or the plain-text program file being loaded by LOAD, CHAIN or MERGE contains a line with more than 255 characters. Attempting to load a text file that has *LF* rather than *CR LF* line endings may cause this error.

24   Device Timeout

The handshake has failed on a serial device or a tape device has reached the end of tape.

25   Device Fault

26   FOR without NEXT

A FOR statement has been encountered for which no matching NEXT statement can be found.

27   Out of paper

An attempt is made to write to a printer which is out of paper or to another parallel device which has raised an out-of-paper condition.

28	*unused*

29   WHILE without WEND

A WHILE statement has been encountered for which no matching WEND statement can be found.

30   WEND without WHILE

A WEND statement has been encountered for which no matching WHILE statement can be found.

31–49	*unused*

50   FIELD overflow

An attempt is made to read, write, or define a FIELD variable beyond the length of the random-access file buffer.

51	**Internal error**
	The `TERM` statement is executed but no terminal manager program has been defined.

52	**Bad file number**
	A file number is accessed to which no file is open, or the file number used in an `OPEN` statement is outside the range of allowable file numbers, or (confusingly) the file specification is empty, malformed or contains illegal characters.

53	**File not found**
	A named file on a disk device cannot be found.

54	**Bad file mode**
	The requested file mode in an `OPEN` statement does not exist or is unsupported for the given device, or the file function called is not supported for this device, or the function or statement called requires a file opened for `RANDOM` and the file is not.

55	**File already open**
	An attempt is made to open a file to a file number that is already in use; or an attempt is made to open a file for `OUTPUT` or `APPEND` on a serial, disk or cassette device when a file (or, on a disk device, a file with the same name) is already open for `OUTPUT` or `APPEND` on that device; or a `KILL` or `NAME` statement is executed on a disk file when a file with the same name is open on the same device.

56	*unused*

57	**Device I/O error**
	An I/O error has occured during input/output to a device. This includes faming errors, CRC check failures and unexpected end-of-tape on cassette devices.

58	**File already exists**
	The proposed new name of a disk file in a `NAME` statement is already in use.

59–60	*unused*

61	**Disk full**
	There is insufficient free space on the disk device to complete the operation.

62	**Input past end**

An attempt is made to retrieve input from a file that has passed its end of file.

| 63 | **Bad record number** |

A random-access file record number is referenced that is outside the permitted range.

| 64 | **Bad file name** |

The file name or other device parameter string in a file specification is malformed or contains illegal characters.

| 65 | *unused* |
| 66 | **Direct statement in file** |

A line with no line number is encountered in a plain-text program file.

| 67 | **Too many files** |
| 68 | **Device Unavailable** |

An attempt is made to access a device that does not exist or is not enabled.

| 69 | **Communication buffer overflow** |

A serial device is receiving more data than fits in its buffer.

| 70 | **Permission Denied** |

The requested access to a file is not granted due to `LOCK` restrictions, operating system locking, or insufficient operating system file permissions.

| 71 | **Disk not Ready** |

The disk device is not ready for access. For example, there is no diskette in a floppy drive or the drive lock is open.

72	**Disk media error**
73	**Advanced Feature**
74	**Rename across disks**

An attempt is made to use the `NAME` statement to move a file from one disk device to another.

| 75 | **Path/File access error** |

An attempt is made to create a directory that already exists or to remove a directory that is not empty.

76	**Path not found**
	An <u>OPEN</u>, <u>MKDIR</u>, <u>RMDIR</u>, or <u>CHDIR</u> statement is executed referring to a (parent) path that does not exist on the disk device.
77	**Deadlock**

Any error code that does not have a message associated to it will generate the message `Unprintable error`.

If an error occurs in direct mode, the error message is printed as above. If the error occurs in a program, the message is supplemented with the line number in which the error occurred. For example,

```
Illegal function call in 100
```

indicates that the illegal function call took place in line number `100`.

If a `Syntax error` occurs during program execution, the error message is followed by a listing of the program line in which the error occurred, wth the cursor positioned at the location where the error was raised.

A `Division by zero` error or, in a floating point calculation, an `Overflow`, will not interrupt execution unless it occurs within an error handling routine. The error message will be printed on the console and the result of the offending calculation will be taken to be the maximum value that fits in the appropriate floating-point variable. `Overflow` in an integer calculation will always interrupt execution like other errors.

Other messages

`Break`

Execution of a compound statement or program has been interrupted by a <u>CONT</u> statement or by a user keyboard interrupt (such as `Ctrl`+`Break`). If the interrupt happens in a program, the `Break` message will be supplemented with the line number in which the interrupt occurred.

`?Redo from start`

The input provided on the console for an <u>INPUT</u> statement does not match the expected format. The number or type of inputs is not correct. Re-enter all inputs.

`Undefined line ref_num in line_num`

The <u>RENUM</u> statement encountered a reference to the line number `ref_num` which is not defined in the program. The reference occurs on line number `line_num`. The undefined line number reference will not be renumbered.

`filename` **Found.**

> A file matching the requested specification has been found on the cassette device. This message only occurs in direct mode.

`filename` **Skipped.**

> A file not matching the requested specification has been encountered on the cassette device. This message only occurs in direct mode.

7. Technical reference

7.1. Tokenised file format

A tokenised program file on a disk device has the following format.

Magic byte

    `FF`

Program lines

Each line is stored as follows:

Bytes	Format	Meaning
2	Unsigned 16-bit little-endian integer.	Memory location of the line following the current one. This is used internally by GW-BASIC but ignored when a program is loaded.
2	Unsigned 16-bit little-endian integer.	The line number.
Variable	Tokenised BASIC, see below.	The contents of the line.
1	`00` (*NUL* byte)	End of line marker.

End of file marker

An `1A` is written to mark the end of file. This is optional; the file will be read without problems if it is omitted.

Tokenised BASIC

The printable ASCII characters in the range `20` — `7E` are used for string literals, comments, variable names, and elements of statement syntax that are not reserved words. Reserved words are represented by their reserved word tokens and numeric literals are represented by numeric token sequences.

Numeric token sequences

Numeric literals are stored in tokenised programs according to the following representation. All numbers are positive; negative numbers are stored simply by preceding the number with `EA`, the token for `-`.

Class	Bytes	Format
Indirect line numbers	3	`0E` followed by an unsigned 16-bit little-endian integer.
Octal integers	3	`0B` followed by an unsigned 16-bit little-endian integer.
Hexadecimal integers	3	`0C` followed by an unsigned 16-bit little-endian integer.
Positive decimal integers less than 11	1	Tokens `11`—`1B` represent 0—10.
Positive decimal integers less than 256	2	`0F` followed by an unsigned 8-bit integer.
Other decimal integers	3	`1C` followed by a two's complement signed 16-bit little-endian integer. GW-BASIC will recognise a negative number encountered this way but it will not store negative numbers itself using the two's complement, but rather by preceding the positive number with `EA`.
Single precision floating-point number	5	`1D` followed by a four-byte single in <u>Microsoft Binary Format</u>.
Double precision floating-point number	9	`1F` followed by an eight-byte double in <u>Microsoft Binary Format</u>.

Keyword tokens

Most keywords in PC-BASIC are *reserved words*. Reserved words are represented in a tokenised program by a single- or double-byte token. The complete list is below.

All function names and operators are reserved words and all statements start with a reserved word (which in the case of `LET` is optional). However, the converse is not true: not all reserved words are statements, functions, or operators. For example, `TO` and `SPC(` only occur as part of a statement syntax. Furthermore, some keywords that form part of statement syntax are not reserved words: examples are `AS`, `BASE`, and `ACCESS`.

Keywords that are not reserved words are spelt out in full text in the tokenised source.

A variable or user-defined function name must not be identical to a reserved word. The list below is an exhaustive list of reserved words that can be used to determine whether a name is legal.

`81`	END	`82`	FOR	`83`	NEXT

84	DATA	85	INPUT	86	DIM
87	READ	88	LET	89	GOTO
8A	RUN	8B	IF	8C	RESTORE
8D	GOSUB	8E	RETURN	8F	REM
90	STOP	91	PRINT	92	CLEAR
93	LIST	94	NEW	95	ON
96	WAIT	97	DEF	98	POKE
99	CONT	9C	OUT	9D	LPRINT
9E	LLIST	A0	WIDTH	A1	ELSE
A2	TRON	A3	TROFF	A4	SWAP
A5	ERASE	A6	EDIT	A7	ERROR
A8	RESUME	A9	DELETE	AA	AUTO
AB	RENUM	AC	DEFSTR	AD	DEFINT
AE	DEFSNG	AF	DEFDBL	B0	LINE
B1	WHILE	B2	WEND	B3	CALL
B7	WRITE	B8	OPTION	B9	RANDOMIZE
BA	OPEN	BB	CLOSE	BC	LOAD
BD	MERGE	BE	SAVE	BF	COLOR
C0	CLS	C1	MOTOR	C2	BSAVE
C3	BLOAD	C4	SOUND	C5	BEEP
C6	PSET	C7	PRESET	C8	SCREEN
C9	KEY	CA	LOCATE	CC	TO
CD	THEN	CE	TAB(	CF	STEP
D0	USR	D1	FN	D2	SPC(
D3	NOT	D4	ERL	D5	ERR
D6	STRING$	D7	USING	D8	INSTR
D9	'	DA	VARPTR	DB	CSRLIN
DC	POINT	DD	OFF	DE	INKEY$
E6	>	E7	=	E8	<
E9	+	EA	-	EB	*
EC	/	ED	^	EE	AND
EF	OR	F0	XOR	F1	EQV
F2	IMP	F3	MOD	F4	\
FD81	CVI	FD82	CVS	FD83	CVD
FD84	MKI$	FD85	MKS$	FD86	MKD$
FD8B	EXTERR	FE81	FILES	FE82	FIELD
FE83	SYSTEM	FE84	NAME	FE85	LSET
FE86	RSET	FE87	KILL	FE88	PUT
FE89	GET	FE8A	RESET	FE8B	COMMON

FE8C	CHAIN	FE8D	DATE$	FE8E	TIME$
FE8F	PAINT	FE90	COM	FE91	CIRCLE
FE92	DRAW	FE93	PLAY	FE94	TIMER
FE95	ERDEV	FE96	IOCTL	FE97	CHDIR
FE98	MKDIR	FE99	RMDIR	FE9A	SHELL
FE9B	ENVIRON	FE9C	VIEW	FE9D	WINDOW
FE9E	PMAP	FE9F	PALETTE	FEA0	LCOPY
FEA1	CALLS	FEA5	PCOPY	FEA7	LOCK
FEA8	UNLOCK	FF81	LEFT$	FF82	RIGHT$
FF83	MID$	FF84	SGN	FF85	INT
FF86	ABS	FF87	SQR	FF88	RND
FF89	SIN	FF8A	LOG	FF8B	EXP
FF8C	COS	FF8D	TAN	FF8E	ATN
FF8F	FRE	FF90	INP	FF91	POS
FF92	LEN	FF93	STR$	FF94	VAL
FF95	ASC	FF96	CHR$	FF97	PEEK
FF98	SPACE$	FF99	OCT$	FF9B	LPOS
FF9A	HEX$	FF9C	CINT	FF9D	CSNG
FF9E	CDBL	FF9F	FIX	FFA0	PEN
FFA1	STICK	FFA2	STRIG	FFA3	EOF
FFA4	LOC	FFA5	LOF		

The following additional reserved words are activated by the option `syntax={pcjr|tandy}` .

FEA4	NOISE	FEA6	TERM

Internal use tokens

The tokens `10` , `1E` and `0D` are known to be used internally by GW-BASIC. They should not appear in a correctly stored tokenised program file.

Microsoft Binary Format

Floating point numbers in GW-BASIC and PC-BASIC are represented in *Microsoft Binary Format (MBF)*, which differs from the IEEE 754 standard used by practically all modern software and hardware. Consequently, binary files generated by either BASIC are fully compatible with each other and with some applications contemporary to GW-BASIC, but not easily interchanged with other software. QBASIC, for example, uses IEEE floats.

MBF differs from IEEE in the position of the sign bit and in using only 8 bits for the exponent, both in single- and in double-precision. This makes the range of allowable numbers in an MBF double-precision number smaller, but their precision higher, than for an IEEE double: an MBF single has 23 bits of precision, while an MBF double has 55 bits of precision. Both have the same range.

Unlike IEEE, the Microsoft Binary Format does not support signed zeroes, subnormal numbers, infinities or not-a-number values.

MBF floating point numbers are represented in bytes as follows:

Single $M_3\ M_2\ M_1\ E_0$

Double $M_7\ M_6\ M_5\ M_4\ M_3\ M_2\ M_1\ E_0$

Here, E_0 is the *exponent byte* and the other bytes form the *mantissa*, in little-endian order so that M_1 is the most significant byte. The most significant bit of M_1 is the *sign bit*, followed by the most significant bits of the mantissa: $M_1 = s_0\ f_1\ f_2\ f_3\ f_4\ f_5\ f_6\ f_7$. The other bytes contain the less-significant mantissa bits: $M_2 = f_8\ f_9\ f_A\ f_B\ f_C\ f_D\ f_E\ f_F$, and so on.

The value of the floating-point number is $v = 0$ if $E_0 = 0$ and $v = (-1)^{s_0} \times mantissa \times 2^{E_0 - 128}$ otherwise, where the mantissa is formed as a binary fraction $mantissa = 0\ .\ 1\ f_1\ f_2\ f_3\ ...$

7.2. Protected file format

The protected format is an encrypted form of the tokenised format. GW-BASIC would refuse to show the source code of such files. This protection scheme could easily be circumvented by changing a flag in memory. Deprotection programs have circulated widely for decades and the decryption algorithm and keys were published in a mathematical magazine.

A protected program file on a disk device has the following format.

Magic byte

`FE`

Payload

Encrypted content of a tokenised program file, including its end of file marker but excluding its magic byte. The encription cipher rotates through an 11-byte and a 13-byte key so that the resulting transformation is the same after 143 bytes. For each byte,

- Subtract the corresponding byte from the 11-byte sequence

  `0B 0A 09 08 07 06 05 04 03 02 01`
- Exclusive-or with the corresponding byte from the 11-byte key

  `1E 1D C4 77 26 97 E0 74 59 88 7C`
- Exclusive-or with the corresponding byte from the 13-byte key

  `A9 84 8D CD 75 83 43 63 24 83 19 F7 9A`
- Add the corresponding byte from the 13-byte sequence

  `0D 0C 0B 0A 09 08 07 06 05 04 03 02 01`

End of file marker

An `1A` is written to mark the end of file. This is optional; the file will be read without problems if it is omitted. Since the end-of-file marker of the tokenised program is included in the encrypted content, a protected file is usually one byte longer than its unprotected equivalent.

7.3. BSAVE file format

A memory-dump file on a disk device has the following format.

Magic byte

`FD`

Header

Bytes	Format	Meaning
2	Unsigned 16-bit little-endian integer.	Segment of the memory block.
2	Unsigned 16-bit little-endian integer.	Offset of the first byte of the memory block.
2	Unsigned 16-bit little-endian integer.	Length of the memory block in bytes.

Payload

The bytes of the memory block.

Footer

On Tandy only, the magic byte and the six bytes of the header are repeated here. This is optional; the file will be read without problems if it is omitted.

End of file marker

An `1A` is written to mark the end of file. This is optional; the file will be read without problems if it is omitted.

7.4. Cassette file format

Files on cassette are stored as frequency-modulated sound. The payload format of files on cassette is the same as for files on disk device, but the headers are different and the files may be split in chunks.

Modulation

A 1-bit is represented by a single 1 ms wave period (1000 Hz). A 0-bit is represented by a single 0.5 ms wave period (2000 Hz).

Byte format

A byte is sent as 8 bits, most significant first. There are no start- or stopbits.

Record format

A file is made up of two or more records. Each record has the following format:

Length	Format	Meaning
256 bytes	All `FF`	2048 ms pilot wave at 1000 Hz, used for calibration.
1 bit	`0`	Synchronisation bit.
1 byte	`16` (*SYN*)	Synchronisation byte.
256 bytes		Data block.
2 bytes	Unsigned 16-bit *big-endian* integer	CRC-16-CCITT checksum.
31 bits	30 1s followed by a 0.	End of record marker.

Tokenised, protected and `BSAVE` files consist of a header record followed by a single record which may contain multiple 256-byte data blocks, each followed by the 2 CRC bytes. Plain text program files and data files consist of a header record followed by multiple single-block records.

Header block format

Bytes	Format	Meaning
1	A5	Header record magic byte
8	8 characters	Filename.
1		File type. 00 for data file, 01 for memory dump, 20 or A0 for protected, 40 for plain text program, 80 for tokenised program.
2	Unsigned 16-bit little-endian integer	Length of next data record, in bytes.
2	Unsigned 16-bit little-endian integer	Segment of memory location.
2	Unsigned 16-bit little-endian integer	Offset of memory location.
1	00	End of header data
239	All 01	Filler

Data block format

Bytes	Format	Meaning
1	8-bit unsigned integer	Number of payload bytes in last record, plus one. If zero, the next record is not the last record.
255		Payload data. If this is the last record, any unused bytes are filled by repeating the last payload byte.

7.5. Emulator file formats

PC-BASIC uses a number of file formats to support its emulation of legacy hardware, which are documented in this section. These file formats are not used by GW-BASIC or contemporary software.

HEX font file format

The HEX file format for bitfonts was developed by Roman Czyborra for the GNU Unifont package. PC-BASIC uses this file format to store its fonts.

A HEX file is an ASCII text file, consisting of lines terminated by `LF`. Each line of this file is one of the following:

- Empty
- A comment, starting with a `#` character.
- A 4 or 6-character hexadecimal Unicode code point, followed by a colon, followed by a 16- or 32-character hexadecimal number representing the glyph. A 16-character number represents a halfwidth (8x16) glyph, with each row of 8 pixels represented by two hexadecimal characters. A 32-character number represents a fullwidth (16x16) glyph, with each row of 16 pixels represented by four hexadecimal characters.

PC-BASIC also encodes 8- and 14-pixel high fonts in this manner; these are encoded as 16-bit high fonts with the remaining rows set to zero.

UCP code page file format

Unicode-codepage mappings are stored in UCP files.

A UCP file is an ASCII text file, consisting of lines terminated by `LF`. Each line of this file is one of the following:

- Empty
- A comment, starting with a `#` character.
- A 2- or 4-character hexadecimal codepage point, followed by a colon, followed by a comma-separated list of 4- or 6-character hexadecimal Unicode code points. If more than one Unicode code point is provided for a codepage point, the code points must combine into a single glyph.

CAS tape file format

A CAS file is a bit-level representation of cassette data introduced by the PCE emulator. CAS-files produced by PC-BASIC start with the characters `PC-BASIC tape`*EOF* . This sequence is followed by seven 0 bits, followed by the tape contents. The seven zero bits are intended to ensure that the tape contents are byte-aligned; the one bit is made up by the synchronisation bit following the pilot wave.

Note that PC-BASIC does not require the introductory sequence to read a CAS-file correctly, nor does it require the contents of a CAS-file to be byte-aligned. However, new files produced by PC-BASIC follow this convention.

7.6. Character codes

Depending on context, PC-BASIC will treat a code point in the control characters range as a control character or as a glyph defined by the active <u>codepage</u> which by default is <u>codepage 437</u>. Code points of `&h80` or higher are always interpreted as a codepage glyph.

ASCII

This is a list of the American Standard Code for Information Interchange (ASCII). ASCII only covers 128 characters and defines the code point ranges `&h00` – `&h1F` and `&h7F` as *control characters* which do not have a printable glyph assigned to them. This includes such values as the *Carriage Return* (`CR`) character that ends a program line.

In the context of this documentation, character `&h1A` (*SUB*) will usually be indicated as `EOF` since it plays the role of end-of-file marker in DOS.

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1_	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2_		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{			}	~	DEL

Codepage 437

This table shows the characters that are produced by the 256 single-byte code points when the DOS Latin USA codepage 437 is loaded, which is the default. Other codepages can be loaded to assign other characters to these code points.

- Code point `&h00` cannot be redefined.
- Redefining characters in the printable ASCII code point range `&h20` – `&h7E` will result in a different glyph being shown on the screen, but the character will continue to be treated as the corresponding ASCII character. It will retain its ASCII value when transcoded into UTF-8. This happens, for example, with the Yen sign (¥) which is assigned to ASCII code point `&h5C` in code page 932: in that

codepage it is treated as if it were a backslash (\ ).

- All other characters can be redefined by loading another <u>codepage</u> with the <u>codepage</u> option. This will affect both the visual glyphs and Unicode character values of those characters.

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
1_	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
2_		!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{			}	~	⌂
8_	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
9_	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
A_	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B_	░	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
C_	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
D_	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	█	▄	▌	▐	▀
E_	α	ß	Γ	π	Σ	σ	µ	τ	Φ	Θ	Ω	δ	∞	φ	ε	∩
F_	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	ⁿ	²	■	

7.7. Keycodes

Scancodes

PC-BASIC uses PC/XT scancodes, which originated on the 83-key IBM Model F keyboard supplied with the IBM PC 5150. The layout of this keyboard was quite distinct from modern standard keyboards with 101 or more keys, but keys on a modern keyboard produce the same scancode as the key with the same function on the Model F. For example, the key that (on a US keyboard) produces the `\` was located next to the left `Shift` key on the Model F keyboard and has scancode `&h2B` . The (US) backslash key still has this scancode, even though it is now usually found above the `Enter` key.

To further complicate matters, keyboards for different locales have their layout remapped in software rather than in hardware, which means that they produce the same scancode as the key that on a US keyboard is in the same location, regardless of which character they actually produce.

Therefore, the `A` on a French keyboard will produce the same scancode as the `Q` on a UK or US keyboard. The aforementioned US `\` key is identified with the key that is generally found to the bottom left of `Enter` on non-US keyboards. For example, on my UK keyboard this is the `#` key. Non-US keyboards have an additional key next to the left `Shift` which on the UK keyboard is the `\`. Therefore, while this key is in the same location and has the same function as the Model F `\`, it has a different scancode.

In the table below, the keys are marked by their function on a US keyboard, but it should be kept in mind that the scancode is linked to the position, not the function, of the key.

Key	Scancode
`Esc`	01
`1 !`	02
`2 @`	03
`3 #`	04
`4 $`	05
`5 %`	06
`6 ^`	07
`7 &`	08
`8 *`	09
`9 (`	0A
`0 )`	0B
`- _`	0C
`= +`	0D
`Backspace`	0E
`Tab`	0F
`q Q`	10
`w W`	11
`e E`	12
`r R`	13
`t T`	14
`y Y`	15
`u U`	16
`i I`	17
`o O`	18
`p P`	19
`[ {`	1A
`] }`	1B
`Enter`	1C
`Ctrl`	1D
`a A`	1E
`s S`	1F
`d D`	20
`f F`	21
`g G`	22

`h H`	23
`j J`	24
`k K`	25
`l L`	26
`; :`	27
`' "`	28
`` ` ~ ``	29
`Left Shift`	2A
`\ |`	2B
`z Z`	2C
`x X`	2D
`c C`	2E
`v V`	2F
`b B`	30
`n N`	31
`m M`	32
`, <`	33
`. >`	34
`/ ?`	35
`Right Shift`	36
`keypad *` `PrtSc`	37
`Alt`	38
`Space`	39
`Caps Lock`	3A
`F1`	3B
`F2`	3C
`F3`	3D
`F4`	3E
`F5`	3F
`F6`	40
`F7`	41
`F8`	42
`F9`	43
`F10`	44
`Num Lock`	45

`Scroll Lock` `Pause`	46
`keypad 7` `Home`	47
`keypad 8` `↑`	48
`keypad 9` `Pg Up`	49
`keypad -`	4A
`keypad 4` `←`	4B
`keypad 5`	4C
`keypad 6` `→`	4D
`keypad +`	4E
`keypad 1` `End`	4F
`keypad 2` `↓`	50
`keypad 3` `Pg Dn`	51
`keypad 0` `Ins`	52
`keypad .` `Del`	53
`SysReq`	54
`\ |` (Non-US 102-key)	56
`F11`	57
`F12`	58
`Left Logo` (Windows 104-key)	5B
`Right Logo` (Windows 104-key)	5C
`Menu` (Windows 104-key)	5D
`ひらがな/カタカナ Hiragana/Katakana` (Japanese 106-key)	70
`\ _` (Japanese 106-key)	73
`?換 Henkan` (Japanese 106-key)	79
`無?換 Muhenkan` (Japanese 106-key)	7B
`半角/全角 Hankaku/Zenkaku` (Japanese 106-key)	29
`¥ |` (Japanese 106-key)	7D
`한자 Hanja` (Korean 103-key)	F1
`한/영 Han/Yeong` (Korean 103-key)	F2
`\ ? °` (Brazilian ABNT2)	73
`keypad .` (Brazilian ABNT2)	7E

e-ASCII codes

Alongside scancodes, most keys also carry a character value the GW-BASIC documentation calls *extended ASCII*. Since this is a rather overloaded term, we shall use the abbreviation *e-ASCII* exclusively for these values. The values returned by the `INKEY$` function are e-ASCII values.

e-ASCII codes are one or two bytes long; single-byte codes are simply ASCII codes whereas double-byte codes consist of a `NUL` character plus a code indicating the key pressed. Some, but certainly not all, of these codes agree with the keys' scancodes.

Unlike scancodes, e-ASCII codes of unmodified keys and those of keys modified by `Shift`, `Ctrl` or `Alt` are all different.

Unmodified, `Shift`ed and `Ctrl`ed e-ASCII codes are connected to a key's meaning, not its location. For example, the e-ASCII for `Ctrl`+`a` are the same on a French and a US keyboard. By contrast, the `Alt`ed codes are connected to the key's location, like scancodes. The US keyboard layout is used in the table below.

Key	e-ASCII	e-ASCII `Shift`	e-ASCII `Ctrl`	e-ASCII `Alt`
`Esc`	1B	1B	1B	
`1 !`	31	21		00 78
`2 @`	32	40	00 03	00 79
`3 #`	33	23		00 7A
`4 $`	34	24		00 7B
`5 %`	35	25		00 7C
`6 ^`	36	5E	1E	00 7D
`7 &`	37	26		00 7E
`8 *`	38	2A		00 7F
`9 (`	39	28		00 80
`0 )`	30	29		00 81
`- _`	2D	5F	1F	00 82
`= +`	3D	2B		00 83
`Backspace`	08	08	7F	00 8C
`Tab`	09	00 0F	00 8D	00 8E
`q Q`	71	51	11	00 10
`w W`	77	57	17	00 11
`e E`	65	45	05	00 12
`r R`	72	52	12	00 13
`t T`	74	54	14	00 14
`y Y`	79	59	19	00 15
`u U`	75	55	15	00 16
`i I`	69	49	09	00 17
`o O`	6F	4F	0F	00 18
`p P`	70	50	10	00 19
`[ {`	5B	7B	1B	
`] }`	5D	7D	1D	
`Enter`	0D	0D	0A	00 8F
`a A`	61	41	01	00 1E
`s S`	73	53	13	00 1F
`d D`	64	44	04	00 20
`f F`	66	46	06	00 21
`g G`	67	47	07	00 22
`h H`	68	48	08	00 23

j J	6A	4A	0A	00 24
k K	6B	4B	0B	00 25
l L	6C	4C	0C	00 26
; :	3B	3A		
' "	27	22		
` ~	60	7E		
\ |	5C	7C	1C	
z Z	7A	5A	1A	00 2C
x X	78	58	18	00 2d
c C	63	43	03	00 2E
v V	76	56	16	00 2F
b B	62	42	02	00 30
n N	6E	4E	0E	00 31
m M	6D	4D	0D	00 32
, <	2C	3C		
. >	2E	3E		
/ ?	2F	3F		
PrtSc			00 72	00 46
Space	20	20	20	00 20
F1	00 3B	00 54	00 5E	00 68
F2	00 3C	00 55	00 5F	00 69
F3	00 3D	00 56	00 60	00 6A
F4	00 3E	00 57	00 61	00 6C
F5	00 3F	00 58	00 62	00 6D
F6	00 40	00 59	00 63	00 6E
F7	00 41	00 5A	00 64	00 6F
F8	00 42	00 5B	00 65	00 70
F9	00 43	00 5C	00 66	00 71
F10	00 44	00 5D	00 67	00 72
F11 (Tandy)	00 98	00 A2	00 AC	00 B6
F12 (Tandy)	00 99	00 A3	00 AD	00 B7
Home	00 47	00 47	00 77	
End	00 4F	00 4F	00 75	
PgUp	00 49	00 49	00 84	
PgDn	00 51	00 51	00 76	

↑	00 48	00 48		
←	00 4B	00 87	00 73	
→	00 4D	00 88	00 74	
↓	00 50	00 50		
keypad 5	35	35		05
Ins	00 52	00 52		
Del	00 53	00 53		

7.8. Memory model

PC-BASIC (rather imperfectly) emulates the memory of real-mode MS-DOS. This means that memory can be addressed in *segments* of 64 KiB. Each memory address is given by the segment value and the `0--65535` byte offset with respect to that segment. Note that segments overlap: the actual memory address is found by `segment*16 + offset`. The maximum memory size that can be addressed by this scheme is thus 1 MiB, which was the size of the *conventional* and *upper* memory in real-mode MS-DOS.

Overview

Areas of memory with a special importance are:

Segment	Name	Purpose
`&h0000`	Low memory	Holds machine information, among other things
`&h13AD` (may vary)	Data segment	Program code, variables, arrays, strings
`&hA000` (EGA)	Video segment	Text and graphics on visible and virtual screens
`&hB000` (MDA)		
`&hB800` (CGA)		
`&hC000`	--	RAM font definition, among other things
`&hF000`	Read-only memory	ROM font definition, among other things

Data segment

The data segment is organised as follows. The addresses may vary depending on the settings of various options; given here are the default values for GW-BASIC 3.23.

Offset	Size (bytes)	Function
`&h0000`	`3429`	Interpreter workarea. Unused in PC-BASIC; can be adjusted with the `--reserved-memory` option.
`&h0D65`	(*max-files*+1) * `322`	File blocks: one for the program plus one for each file allowed by `--max-files`.
`&h126D`	`3 +` *c*	Program code. An empty program uses 3 bytes.
`&h1270 +` *c*	*v*	Scalar variables.
`&h1270 +` *c + v*	*a*	Array variables.
`&hFDFC -` *s*	*a*	String variables, filled downward from `&hFDFC`
`&hFDFC`	`512`	BASIC stack, size set by `CLEAR` statement.
`&hFFFE`		Top of data segment, set by `CLEAR` statement.

8. Developer's guide

The features described in this guide are intended for Python developers only. They are experimental, may not work as expected, and may be removed from future releases without warning. You may not be able to get help if you have any problems. Luckily, none of the features described here are needed for the normal functioning of PC-BASIC.

8.1. Session API

PC-BASIC can be loaded as a package from Python, which makes it possible to call BASIC code directly from Python.

class `Session(**kwargs)`

Open a PC-BASIC session. The session object holds the interpreter state, e.g. the value of variables, program code and pointers, screen state, etc. Note that `Session` can be used as a context manager with the `with` statement.

Keyword arguments are largely (but not entirely) analogous to PC-BASIC command-line options.

By default, the Session object grabs the standard input and output as keyboard an screen. This may be undesirable in some applications; in such cases, set the keyword arguments `input_streams` and `output_streams` explicitly (for example, to `None`).

execute(*basic_code*)

Execute BASIC code. `basic_code` can be commands or program lines, separated by `\n` or `\r`.

evaluate(*basic_expr*)

Evaluate a BASIC expression and return its value as a Python value. For type converson rules, see get_variable.

set_variable(*name, value*)

Set the value of a scalar or array to a Python value.

`name` is a valid BASIC name, including the sigil, and is not case-sensitive. If the target is an array, `name` should end with `()`.

`value` should be of a compatible type: `int`, `bool` or `float` for numeric variables and `bytes` or `unicode` for strings. If the target is an array, `value` should be a `list` of such values. Multi-dimensional arrays should be specified as nested `list`s.

`bool`s will be represented as in BASIC, with `-1` for `True`. `unicode` will be converted according to the active codepage.

get_variable(*name*)

Retrieve the value of a scalar or array as a Python value.

`name` is a valid BASIC name, including the sigil, and is not case-sensitive. If the target is an array, `name` should end with `()` .

Integers will be returned as `int` , single- and double-precision values as `float` , and string as `bytes` . If the target is an array, the function returns a (nested) `list` of such values.

close()

Close the session: closes all open files and exits PC-BASIC. If used as a context manager, this method is called automatically.

8.2. Extensions

It's possible to enable your own BASIC statements using *extensions*. An extension is a Python object or module loaded through the `--extension` option or through the `extension` parameter of the `Session` object.

Python functions and other callable objects in the extension's namespace will be made accessible through basic as extension statements or functions whose name starts with an underscore `_`

In order for this to work, the function must have a name that is also a valid BASIC variable name: alphanumeric only, no underscores, not equal to a BASIC keyword. The name will be case-insensitive in BASIC; that is, `def mytestfunc(): print 1` and `def myTestFunc(): print 2` both map to the extension statement or function `_MYTESTFUNC`. Which one of these functions would be chosen is not defined, so avoid this situation.

Any arguments provided to the extension statement or function are supplied to the Python function as the corresponding type: BASIC integers become `int` s, single- and double-precision numbers become `float` s and strings become `bytes` (*not* `unicode` and no codepage conversions are applied).

For example, a call to `_MYTESTFUNC 5, "test-string"` would expect to find a Python function `mytestfunc(i, s)` with two parameters, and will supply `i=int(5)` and `a=bytes('test-string')`.

The same Python function can also be called as an extension function, e.g. `A = _MYTESTFUNC(5, "test-string")`. If called as a function, `mytestfunc(i, s)` must return a value that is one of `int`, `float`, both of which will be converted to a BASIC double-precision float; `bool`, which will be converted to a BASIC integer; or `bytes` or `unicode`, which will be converted to a BASIC string.

8.3. Examples

```
import pcbasic

import random


with pcbasic.Session(extension=random) as s:

    s.execute('a=1')

    print s.evaluate('string$(a+2, "@")')

    s.set_variable('B$', 'abcd')

    s.execute('''

        10 a=5

        20 print a

        run

        _seed(42)

        b = _uniform(a, 25.6)

        print a, b

    ''')
```

9. Acknowledgements

9.1. Contributors

PC-BASIC would not exist without those contributing code, reporting bugs, sending in patches, and documenting GW-BASIC's behaviour. Thank you all!

Development is led by

- **Rob Hagemans**

Bug fixes and guidance by

- **Wengier Wu**
- **Jan Bredenbeek**
- **WJB**

Avid testers and bug hunters

- **Ronald Herrera**
- **Kenneth Wayne Boyd**
- **Nauman Umer**
- **Steve Pagliarulo**
- **Daniel Santana**
- **Miguel Dorta**
- **Patrik**
- **Duane**

9.2. Shoulders of Giants

PC-BASIC incorporates code derived from other projects, in particular:

- **J-L Morel**'s Win32::Console::ANSI
- **Valentin Lab**'s win_subprocess.py
- **Marcus von Appen**'s PySDL2

PC-BASIC depends on the following open-source projects:

- Python
- Setuptools
- NumPy
- PySerial
- PyParallel
- Simple DirectMedia Layer (SDL)
- SDL2_gfx
- PyGame
- PyAudio

9.3. Fond memories

PC-BASIC would not have been what it is without the following open-source projects which it has depended on in the past:

- **Tom Rothamel**'s <u>PyGame Subset for Android</u> (superseded by <u>RAPT</u>)
- <u>Python for Windows Extensions (PyWin32)</u>
- <u>PExpect</u>

9.4. Technical Documentation

Building PC-BASIC would have been impossible without the immense amounts of technical documentation that has been made available online. It has proven not to be feasible to compile a complete list of the documentation used. Many thanks to all those who make technical information freely available, and apologies to those whose contribution I have failed to acknowledge here.

GW-BASIC tokenised file format
- **Norman De Forest's** seminal documentation of GW-BASIC tokens. *This documentation was the starting point for the development of PC-BASIC.*
- **Dan Vanderkam's** online GW-BASIC decoder

GW-BASIC protected file format
- **Paul Kocher**, *The Cryptogram computer supplement* **19**, American Cryptogram Association, Summer 1994

Technical information on many topics
- VOGONS
- **Erik S. Klein's** vintage computer forums
- **John Elliott's** Vintage PC pages
- **Peter Berg's** Pete's QBasic/QuickBasic site
- **Vernon Brooks's** PC-DOS retro

Video hardware
- **Dan Rollins'** TechHelp pages on PC video memory layout
- **Great Hierophant's** Nerdly Pleasures Blog

Microsoft Binary Format
- Forum contributions by **Julian Brucknall** and **Adam Burgoyne**

Data cassette format
- **Mike Brutman's** Analysis of the IBM PC data cassette format
- **Dan Tobias'** IBM PC data cassette format documentation

Serial ports
- **Craig Peacock's** documentation on interfacing the serial port
- **Christopher E. Strangio's** tutorial on the RS232 standard
- QB64 documentation

Keyboard scancodes

- **John Savard**'s Scan Codes Demystified
- **Andries Brouwer**'s extensive reference of Keyboard scancodes
- **Philip Storr**'s PC Hardware book
- Altek Instruments documentation on PC Keyboard Scan Codes

9.5. Fonts

- **Henrique Peron**'s CPIDOS codepage pack
- **Dmitry Bolkhovityanov**'s Uni-VGA font
- **Roman Czyborra**, **Qianqian Fang** and others' GNU UniFont
- DOSBox VGA fonts
- **Andries Brouwer**'s CPI font file format documentation

9.6. Unicode-codepage mappings

- The Unicode Consortium and contributors
- GNU libiconv Project
- Aivosto
- **Konstantinos Kostis'** Charsets Index
- IBM CDRA
- **Masaki Tojo**'s Camellia

9.7. Bibliography

- *GW-BASIC 3.23 User's Guide*, Microsoft Corporation, 1987.
- *IBM Personal Computer Hardware Reference Library: BASIC*, IBM, 1982.
- *Tandy 1000 BASIC, A Reference Guide*, Tandy Corporation.
- **William Barden, Jr.**, *Graphics and Sound for the Tandy 1000s and PC Compatibles*, Microtrend, 1987.
- **Don Inman** and **Bob Albrecht**, *The GW-BASIC Reference*, Osborne McGraw-Hill, 1990.
- **Thomas C. McIntyre**, *BLUE: BASIC Language User Essay*, 1991, available online.
- **David I. Schneider**, *Handbook of BASIC: Third Edition for the IBM PC, XT, AT, PS/2, and Compatibles*, Brady, 1988.

9.8. Development tools

Development tools:

- Git
- Atom
- PyLint
- Coverage
- Ubuntu
- The GNU Base System

Documentation tools:

- LXML
- Markdown
- Prince

Packaging tools:

- Wheel
- Twine
- cx_Freeze
- fpm

Source code and releases are hosted on:

- GitHub
- SourceForge
- PyPI

9.9. Emulators

These excellent emulators have been indispensable tools in documenting the behaviour of various Microsoft BASIC dialects.

- DOSBox
- MESS
- PCE PC Emulator

Licences

PC-BASIC interpreter

Copyright © 2013—2018 Rob Hagemans.

Source code available at https://github.com/robhagemans/pcbasic.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

PC-BASIC documentation