BASLOAD-ROM User Manual

May 3, 2025

Contents

1 Introduction **2**

2 Hello World Example **2**

3 Source Code Files **3**

4 BASIC Syntax **3**

4.1 General . . . 3

4.2 Identifiers . . . 4

4.3 Labels . . . 4

4.4 Variables . . . 5

4.5 Whitespace to Separate Identifiers . . . 5

5 BASLOAD Options **5**

5.1 General . . . 5

5.2 Option: ## Comment . . . 6

5.3 Option: #REM 0 | 1 . . . 7

5.4 Option: #INCLUDE "filename" . . . 7

5.5 Option: #AUTONUM <int8 > . . . 7

5.6 Option: #CONTROLCODES 0 | 1 . . . 8

5.7 Option: #SYMFILE "@:filename" . . . 8

5.8 Option: #SAVEAS "@:filename" . . . 8

5.9 Option: #MAXCOLUMN <int8> . . . 9

5.10 Option: #DEFINE <identifier name> <int16> . . . 9

5.11 Conditional Translation of Source Code . . . 9

5.12 Option: #TOKEN <identifier name> <int16> . . . 10

6 Running BASLOAD **10**

6.1 From BASIC 10

6.2 API 10

A Named PETSCII Control Characters **12**

1 Introduction

BASLOAD runs natively on the Commander X16 retro computer. It loads BASIC source code in plain text format from the SD card and converts it to runnable BASIC programs.

The goal of BASLOAD is to make BASIC programming for the Commander X16 more convenient.

The main benefits of BASLOAD are:

- Source code may be edited in any text editor.
- Line numbers are not used in the source code; named labels are decleared as targets for GOTOs and GOSUBs.
- Long variable names are supported, while only the first two characters of a variable name are significant in the built-in BASIC.

This manual describes the ROM based version of BASLOAD. The original RAM based version of BASLOAD is depreciated.

The Commander X16 was devised by David Murray a.k.a. the 8-Bit Guy. For more information on the platform, go to www.commanderx16.com.

2 Hello World Example

Let's begin with a simple Hello world example in BASLOAD formatted source code. Open your text editor, type in the code below, and save it to the file "HELLO.BAS" on the SD card.

```
LOOP:
    PRINT "HELLO, WORLD!"
    GOTO LOOP
```

Convert the file to a runnable program with the following command:

```
BASLOAD "HELLO.BAS"
```

The program is now loaded into memory. You can type LIST to show it and RUN to execute it.

If you get a ?Syntax Error, it means that BASLOAD is not present in your ROM.

3 Source Code Files

BASLOAD expects source code files to be stored on the Commander X16's SD card.

Source code files may be created with any text editor as long as the following requirements are met:

- The source code must be plain ASCII or PETSCII encoded text.
- Line breaks may be encoded with CR, LF or CRLF.
- A line may not be more than 250 characters long.

4 BASIC Syntax

4.1 General

Source code written for BASLOAD is meant to be as close as possible to BASIC typed in at the on-screen editor (the "built-in BASIC").

The source code follows the same syntax and supports the same commands and operators as the built-in BASIC. For further information on this topic, see x16-docs/Basic Programming.

There are, however, four important differences between BASLOAD source code and the built-in BASIC:

- Line numbers are not used in BASLOAD source code.
- Labels are decleared as targets for GOTOs, GOSUBs and other commands that normally take a line number.
- Variable names have up to 64 significant characters, compared to two in the built-in BASIC.
- Some whitespace is required to separate identifiers, *i.e.* commands, labels, and variables, from each other.

4.2 Identifiers

BASIC commands, labels, variables and defined constants are commonly referred to as "identifiers" in this manual.

Identifiers start with a letter, any of A–Z.

Subsequent characters may be any of:

- A letter A–Z.
- A digit 0–9.
- An underscore.
- A period (.).

Identifiers are not case-sensitive, and may be at most 64 characters long.

An identifier may not be exactly the same as an already existing identifier.

4.3 Labels

Labels are used as targets for commands that take a line number in the built-in BASIC, such as GOTO, GOSUB and RESTORE.

A label is decleared at the beginnig of a line. There may, however, be whitespace before the start of the declaration.

A label must be a valid identifier, as described above. The end of a label declaration is marked with a colon.

Examples:

- LOOP: is a valid label declaration
- PRINT: is not a valid label declaration as PRINT is a reserved word
- PRINTME: is valid

When you want to refer to a label later on in the source code, for instance after a GOTO command, you just type the label name without colon.

Example:

```
LOOP:
    ...
    GOTO LOOP
```

4.4 Variables

Variables are automatically decleared when they are first used in code.

A variable must be a valid identifier as described above.

As in the built-in BASIC, you add a dollar sign ($) to the end of the variable name to make it a string or a percentage sign (%) to make it an integer.

Examples:

```
INPUT "WHAT IS YOUR NAME"; NAME$
PRINT "HELLO, "; NAME$
```

4.5 Whitespace to Separate Identifiers

BASLOAD requires some whitespace to separate identifiers. Specifically, two identifiers next to each other must be separated by whitespace if not otherwise separated by a character outside the group of characters allowed in identifier names.

The following characters are recognized as whitespace:

- Blank space (PETSCII/ASCII 32)
- Horizontal tab (PETSCII/ASCII 9)
- Shift + Blank space (PETSCII/ASCII 160)

Examples:

- PRINTME needs to be separated if you want to PRINT the value of ME
- PRINT"ME" does not need to be separated as double quote cannot be part of an identifier name

5 BASLOAD Options

5.1 General

BASLOAD lets you put options in the source code that affect the output, similar to directives in C/C++.

An option must be placed at the beginning of a line, but there may be whitespace before it.

Options have the following general syntax:

```
#<NAME> <ARGUMENT 1> ... <ARGUMENT n>
```

The following options are supported:

- ##
- #REM
- #INCLUDE
- #AUTONUM
- #CONTROLCODES
- #SYMFILE
- #SAVEAS
- #MAXCOLUMN
- #DEFINE
- #IFDEF
- #IFNDEF
- #ENDIF
- #TOKEN

Arguments are separated by one or more whitespace characters. Arguments may be integer values in decimal format, identifier names or strings. A string may optionally be enclosed in duoble quotes, which makes it possible for it to contain whitespace characters.

5.2 Option: ## Comment

This option is an alternative comment that is never outputted to runnable code.

Example:

```
## A comment
```

5.3 Option: #REM 0 | 1

This option lets you select whether REM statements are included in the resulting code or not. #REM 0 turns off the output and #REM 1 turns it on again.

It is possible to change the option value multiple times in the source code. The option takes effect from the line where it is encountered and remains in force until changed.

The default value is 0 (off).

Example that turns on output of REM statements:

```
#REM 1
```

5.4 Option: #INCLUDE "filename"

This option includes the content of another BASIC source file where it is encountered.

An included source file can in its turn include another source file. The maximum depth of includes is limited by the fact that the Commander X16 can have at most ten files open at the same time.

Example that includes the file FUNC.BAS:

```
#INCLUDE "FUNC.BAS"
```

5.5 Option: #AUTONUM <int8 >

The autonum option makes it possible to set how many steps the line number of the resulting code is advanced for each outputted line.

The option takes effect from the line where it is found and remains in force until changed. It is possible to change the value multiple times in a source file.

The default step value is 1.

This option may come in handy if you want to make room to insert code directly into the runnable code, for instance for debugging.

Example that advances the output line counter 10 steps per line

```
#AUTONUM 10
```

5.6 Option: #CONTROLCODES 0 | 1

The controlcodes option makes it possible to type named PETSCII control characters, such as arrow up or down.

The available named control characters are listed in Appendix A.

The controlcodes option takes effect from the line where it is encountered and remains in force until changed. It is possible to set and change the option multiple times in the source code.

The default value is 0 (off).

The named control characters are only available within a string.

If you want to put a left curly bracket or a backslash in a string, you need to escape each of them with a backslash while this option is active.

Examples:

```
#CONTROLCODES 1
PRINT "{CLEAR}Hello, world": REM {CLEAR}->PETSCII $93
PRINT "\{CLEAR} clears the screen": REM {CLEAR} unconverted
```

5.7 Option: #SYMFILE "@:filename"

This option writes symbols (labels and variables) found during the translation of the source code to the specified symbol file.

Symbol files are inteded to help while debugging BASLOAD code.

The symfile option may only be placed at the top of the source code before any runnable code has been outputted.

The option may not be used more than once in the source code.

It is recommended that you add @: before the filename. That will cause an existing symbol file to be overwritten, which generally is what you want. Otherwise it is not possible to run BASLOAD multiple times without file exists error.

Example that writes the symbol file "MYPRG.SYM", overwriting the file if it exists:

```
#SYMFILE "@:MYPRG.SYM"
```

5.8 Option: #SAVEAS "@:filename"

This option autosaves the tokenized program to the specified file name. Preprend the file name by @: if you always want to overwrite an existing file.

5.9 Option: #MAXCOLUMN <int8>

The MAXCOLUMN options sets the maximum line width of the source file. If exceeded, BASLOAD stops with an error.

The default value is 250 characters.

5.10 Option: #DEFINE <identifier name> <int16>

This option defines a 16 bit constant integer.

It is possible to redefine the constant multiple times. It is, however, not possible to use the same identifier name as is also used for BASIC commands, variables or labels.

After it has been defined, the name of a constant can be used in the source code. It is then replaced by its integer value in the resulting runnable code.

Defining constants is also useful for conditional translation of the source code, which is discussed more in depth below.

Example:

```
#DEFINE MYPARAM 1
PRINT MYPARAM: REM TRANSLATES TO PRINT 1
```

5.11 Conditional Translation of Source Code

BASLOAD supports conditional translation of the source code using the #IFDEF, #IFNDEF and #ENDIF options.

IFDEF checks if the specified defined constant exists. Other identifiers, such as BASIC commands, variables and labels, are ignored by IFDEF. If the defined constant does not exist, all subsequent code in the source file is ignored until the matching ENDIF statement.

IFNDEF works in the opposite way to IFDEF, including subsequent code if the the defined constant is not defined.

It is possible to nest IFDEF and IFNDEF statements. The maximum level of nested statements is 16.

Example:

```
#DEFINE MYPARAM 1
#IFDEF MYPARAM
    PRINT "HELLO": REM INCLUDED IN TRANSPILED CODE
#ENDIF
```

```
#IFNDEF MYPARAM
    PRINT "WORLD": REM SUPPRESSED, NOT INCLUDED IN TRANSPILED CODE
#ENDIF
```

5.12 Option: #TOKEN <identifier name> <int16>

The TOKEN option inserts a command or operator into the symbol table. It can be used to create an alias of BASIC commands or to insert any raw bytes into the resulting code.

Example that creates a token for the PI character:

```
#TOKEN PI 255
PRINT PI
```

6 Running BASLOAD

6.1 From BASIC

There are two ways to start BASLOAD.

The first method is to use the BASLOAD command available in X16 BASIC. The name of the source file is specified within double quotes, for example:

```
BASLOAD "MYPROGRAM.BAS"
```

The second method is a keyboard shortcut in X16 Edit, the built-in text editor.

6.2 API

BASLOAD can be integrated into other programs, and started through its API as set out below.

Call address: $C000

Input parameters:

Register	Address	Description
R0L	$02	File name length

R0H	$03	Device number
—	$00:BF00–BFFF	File name buffer

Returns:

Register	Address	Description
R1L	$04	Return code
R1H..R2H	$05–07	Source line number where error occured
R3	$08–09	Pointer to source file name where error occured (always bank 1)
—	$00:BF00–BFFF	Plain text return message

The possible return codes are:

- 0 = OK
- 1 = Line too long
- 2 = Symbol too long
- 3 = Duplicate sybols
- 4 = Symbol table full
- 5 = Out of variable names
- 6 = Label expected
- 7 = Label not expected
- 8 = Line number overflow
- 9 = Option unknown
- 10 = File error
- 11 = Invalid param
- 12 = Invalid control code
- 13 = Invalid symbol file
- 14 = Symbol file I/O error
- 15 = File name not specified
- 16 = BASIC RAM full
- 17 = Symbol not in scope
- 18 = Too many nested IFs
- 19 = ENDIF without IF

A Named PETSCII Control Characters

In order to use named control characters you must first put this line in the source code:

```
#CONTROLCODES 1
```

PETSCII	Name	Description
$01	{SWAP COLORS}	SWAP COLORS
$02	{PAGE DOWN}	PAGE DOWN
$03	{STOP}	STOP
$04	{END}	END
$05	{WHITE}	COLOR: WHITE
$05	{WHT}	COLOR: WHITE
$06	{MENU}	MENU
$07	{BELL}	BELL
$08	{CHARSET SWITCH OFF}	DISALLOW CHARSET SW (SHIFT+ALT)
$09	{TAB}	TAB / ALLOW CHARSET SW
$09	{CHARSET SWITCH ON}	TAB / ALLOW CHARSET SW
$0A	{LF}	LF
$0D	{CR}	RETURN
$0E	{LOWER}	CHARSET: LOWER/UPPER
$0F	{ISO ON}	CHARSET: ISO ON
$10	{F9}	F9
$11	{DOWN}	CURSOR: DOWN
$12	{RVS ON}	REVERSE ON
$13	{HOME}	HOME
$14	{BACKSPACE}	DEL (PS/2 BACKSPACE)
$15	{F10}	F10
$16	{F11}	F11
$17	{F12}	F12
$18	{SHIFT TAB}	SHIFT+TAB
$19	{DEL}	FWD DEL (PS/2 DEL)
$1B	{ESC}	ESC
$1C	{RED}	COLOR: RED
$1D	{RIGHT}	CURSOR: RIGHT
$1E	{GREEN}	COLOR: GREEN
$1E	{GRN}	COLOR: GREEN
$1F	{BLUE}	COLOR: BLUE
$1F	{BLU}	COLOR: BLUE
$80	{VERBATIM}	VERBATIM MODE
$81	{ORANGE}	COLOR: ORANGE

$81	{ORG}	COLOR: ORANGE
$82	{PAGE UP}	PAGE UP
$85	{F1}	F1
$86	{F3}	F3
$87	{F5}	F5
$88	{F7}	F7
$89	{F2}	F2
$8A	{F4}	F4
$8B	{F6}	F6
$8C	{F8}	F8
$8D	{SHIFT CR}	SHIFTED RETURN
$8E	{UPPER}	CHARSET: UPPER/PETSCII
$8F	{ISO OFF}	CHARSET: ISO OFF
$90	{BLACK}	COLOR: BLACK
$90	{BLK}	COLOR: BLACK
$91	{UP}	CURSOR: UP
$92	{RVS OFF}	REVERSE OFF
$93	{CLEAR}	CLEAR
$93	{CLR}	CLEAR
$94	{INSERT}	INSERT
$95	{BROWN}	COLOR: BROWN
$96	{LIGHT RED}	COLOR: LIGHT RED
$97	{GREY 3}	COLOR: DARK GRAY
$98	{GREY 2}	COLOR: MIDDLE GRAY
$99	{LIGHT GREEN}	COLOR: LIGHT GREEN
$9A	{LIGHT BLUE}	COLOR: LIGHT BLUE
$9B	{GREY 1}	COLOR: LIGHT GRAY
$9C	{PURPLE}	COLOR: PURPLE
$9C	{PUR}	COLOR: PURPLE
$9D	{LEFT}	CURSOR: LEFT
$9E	{YELLOW}	COLOR: YELLOW
$9E	{YEL}	COLOR: YELLOW
$9F	{CYAN}	COLOR: CYAN
$9F	{CYN}	COLOR: CYAN