Syntax

Introduction

This page is about the ZX BASIC language syntax. It is part of the Language Reference Guide. ZX BASIC aims to be a modern BASIC dialect but tries to keep some (many) of the original Sinclair BASIC features for the nostalgic. So you can use it in both ways.

The BASIC dialect is mainly based in FreeBasic. Many of the language specifications have been taken from there.

ZX BASIC syntax overview

If you have ever programmed in legacy BASIC (either Sinclair BASIC or any other flavour) you will already know that BASIC languages are **line oriented**. Each sentence or group of sentences are separated in lines (ended with a carriage return).

Nowadays this is not necessary, but ZX BASIC allows you to use lines and line numbers for compatibility (and nostalgic!) reasons:

```
10 REM This is a comment.
20 PRINT "Hello world!"
```

Other than that, line numbers are ignored (well, not exactly: line numbers are treated as labels). So the previous BASIC program could be also written this way:

```
REM This is a comment.
PRINT "Hello world!"
```

Lines and sentences

Since ZX BASIC is **line oriented** this implies that the "end of line" (also known as "carriage return" or "") is taken into account during syntax checking, and you cannot break lines in the middle of a sentence:

```
REM The following line has a syntax error
PRINT
    "Hello world!"
```

Other languages (like C or Pascal) allows this because they're not line oriented. If you need to break a line, use the underline character (_) at the end of line to tell the compiler the current line continues in the next one:

```
REM The following line is broken into two, but it's ok
PRINT _
    "Hello world!"
```

(Notice the _ character at the end of the second line).

Sentences and block of sentences

A sentence is the simplest BASIC instruction (e.g. PRINT). Sentences might contain "arguments" and can be separated by a "colon" (:) as in Sinclair BASIC or by "end of line". A "block of sentences" are just a group of sentences one after another. Usually the reserved word **END** denotes the end of such block. E.g.

```
IF a > b THEN
    PRINT "A is greater than B"
    PRINT "and that's all"
END IF
```

In the previous example, everything between **THEN** and **END IF** conforms a "block of sentences". Some sentences (like the shown **IF**) works with sentences block. They are called "compound sentences".

Identifiers

Identifiers are used to denote variables, labels, sentences and functions. Some identifiers are *reserved* for ZX BASIC statements (e.g. PRINT) or predefined functions (e.g. **COS**). Proceed to the identifiers page for a list of *reserved words*.

Numbers

Decimal numbers should be entered as one would normally expect. The compiler also accepts hexadecimal and binary numbers.

- For hexadecimal numbers, use either a trailing h (e.g. `9Ch`) or a leading dollar symbol (`$9C`).
- For binary numbers, use either a trailing b (e.g. `11001001b`) or a leading percent symbol (`%11001001`)

```
Note: Take care of using a trailing h for hexadecimals starting with a letter.
E.g. C9h is an identifier!
```

v: latest

When writing an hexadecimal number with a trailing h, if the number begins with a letter (e.g. C9h), prefix it with a 0 digit. So, `C9` hex should be written as `0C9h` or `$C9`.

Comments

As shown in the previous examples, the "reserved word" **REM** is used for comments and "remarks". However, you can also use the single quote (') character for comments instead of **REM**:

```
10 REM This is a comment
20 'This is also a comment
30 PRINT "Hello world!"
```

You can also comment a block of lines, and even comment *broken* lines. Read comments article for more information.

Graphic characters

The ZX Spectrum had two types of graphics characters; block graphics and user-defined graphics (UDG). The method for entering them into ZX BASIC is the same as that found in the .bas file format created by Paul Dunn for his BASin BASIC IDE.

UDG can be entered into the code with an escape before the letter that corresponds to the udg. For the first udg, for example, use \A

Block graphics characters can be entered into code with a similar escape sequence. The \ is used to escape the characters and a combination of the characters `:` (colon), `'` (apostrophe) and `.` (dot) are used to represent the blocks. So a full solid block would be `\::` An "L" shaped block would be `\:.` and an "r" shaped one would be `\:'`. The system is fairly intuitive when you see how it works. To put it another way, a block graphic is an escape (\ ) followed by two characters. The `:` represents both top and bottom as ink; `'` represents top only, and `.` represents bottom only. A blank space represents both blocks blank or paper. The complete list of possibilities is this:

```
\·· (Space, Space)              CHR$(128)
\·' (Space, Apostrophe)         CHR$(129)
\'· (Apostrophe, Space)         CHR$(130)
\'' (Apostrophe, Apostrophe)    CHR$(131)
\·. (Space, Period)             CHR$(132)
\·: (Space, Colon)              CHR$(133)
\'. (Apostrophe, Period)        CHR$(134)
\': (Apostrophe, Period)        CHR$(135)
\.· (Period, Space)             CHR$(136)
\.' (Period, Apostrophe)        CHR$(137)
\:· (Colon, Space)              CHR$(138)
\:' (Colon, Apostrophe)         CHR$(139)
\.. (Period, Period)            CHR$(140)
\.: (Period, Colon)             CHR$(141)
\:. (Colon, Period)             CHR$(142)
\:: (Colon, Colon)              CHR$(143)
```

v: latest

Other escaped characters

```
\\ The \ backslash symbol.
\` The £ pound sterling symbol. (Just backtick ` works too)
\#nnn Any character, where nnn is a decimal number in the range 000 to 255.
\* The (C) Copyright Symbol.
```

Embedded color control codes

Sometimes, in a program, one might wish to embed colour control codes into strings for printing. This is possible using the same schema as Paul Dunn's BASIC IDE BASin.

The escape sequences for control characters are as follows:

- `\{iN}` Ink colour N, where N is in the range 0 to 8.
- `\{pN}` Paper colour N, where N is in the range 0 to 8.
- `\{bN}` Bright N, where N is 0 or 1.
- `\{fN}` Flash N, where N is 0 or 1.

So, for example, an embedded control code for red ink would be `\{i2}`. 8 is used to signify "transparent" (i.e. do not change the ink/paper value in the square being printed)

Data types

ZX Basic types ranges from 8 to 32 bits for integer formats. It also supports floating point format (the ZX ROM 40 bits floating point from the ROM FP Calculator) and "Fixed" for fixed point arithmetic. See types page for more information.

Inline assembly

The Compiler supports inline assembly, starting with the ASM directive and ending with an END ASM directive. Between these two, raw z80 assembly becomes legal. This assembly data will be passed directly to the assembler as part of the compiled assembler source.

Note that the rules for assembly change dramatically from standard ZX BASIC, and this mode is not for the unwary.

Of note

- Comments are begun with a semicolon (`;`) instead of an apostrophe (`'`).
- The assembler supports `DEFB` to define comma separated bytes, or a quote delimited s v: latest
- The assembler supports `DEFS n,B` to shortcut a series of n bytes of value B.
- The assembler supports it's own set of #directives, including incbin

A novice user of assembly would be well advised to examine code held in the library for examples and usage.

v: latest