LIQUID BASIC 3

Quick Reference Guide

Version 3.07

Written by Bryan Flick

Introduction

Liquid BASIC 3 is a modern BASIC (Beginner's All-purpose Symbolic Instruction Code) dialect. It includes all the standard BASIC commands, plus support for structured programming, functions, files, databases, and artificial intelligence. It also includes a complete graphics and sound engine.

Getting Started

Liquid BASIC runs in a 1280x800 window. Once Liquid BASIC is initialized, it will display a "Ready." prompt. Enter HOME to load and run the included Home.bas program. The program provided has a simple menu interface to browse and run the demonstration programs. Use the up/down arrow keys to move the selection up or down. Press Enter to run the selected demonstration (if it is a BASIC program, printed in white) or change to the selected directory (if it's a directory, printed in green).

Commands can be entered and executed immediately in Direct Mode. If the line starts with a line number instead, then the current line is inserted into the program currently in memory. Valid line numbers can be from 0 to 65535. Line numbers are not used for branching; they are simply used to sort the program while editing and are not saved to disk.

Press the CTRL + Home keys together to reset the user interface (soft reset).

The Line Editor

The Liquid BASIC line editor is similar to the Commodore 64 editor. Lines are "physical" (a literal line of characters) and "logical" (multiple physical lines grouped together to read as one). Typing to the end of a physical line will insert a new physical line to create one long logical line.

The arrow keys can be used to move the cursor around.

Use the Home key to go to the beginning of a logical line. Use the End key to go to the end of a logical line.

Use the SHIFT + Home keys together to move the cursor to the "home" position (upper left corner of the screen).

Use the Backspace key to delete the character to the left of the cursor on the current logical line. Use the Delete key to delete the character under the cursor on the current logical line. Use the Insert key to toggle between insert (half cursor) and overwrite (full cursor) mode.

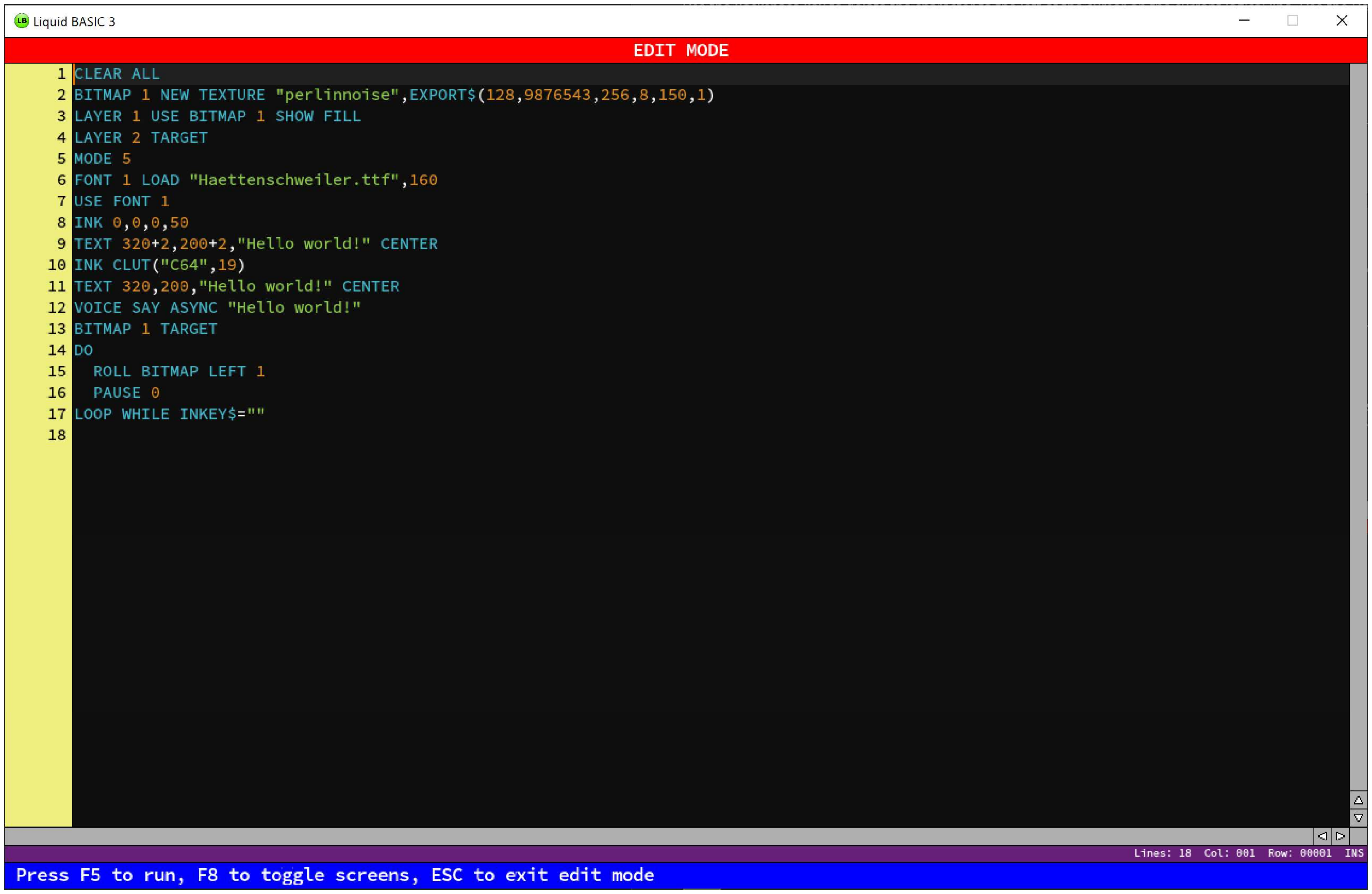
The Text Editor

Liquid BASIC also includes a modern way to edit BASIC programs using a full screen text editor. Enter EDIT to open the text editor.

Text can be selected using the mouse or the SHIFT key. Use CTRL + C to copy the selected text, CTRL + X to cut, and CTRL + V to paste.

Use CTRL + Z to undo an operation and CTRL + Y to redo it.

Press F5 to run the program. Press F8 to toggle between the text editor and the Liquid BASIC screen. Press Esc to exit the full screen editor and place the editor contents back into program memory (renumbered starting at line number 10 with an increment of 10).



```
CLEAR ALL
BITMAP 1 NEW TEXTURE "perlinnoise",EXPORT$(128,9876543,256,8,150,1)
LAYER 1 USE BITMAP 1 SHOW FILL
LAYER 2 TARGET
MODE 5
FONT 1 LOAD "Haettenschweiler.ttf",160
USE FONT 1
INK 0,0,0,50
TEXT 320+2,200+2,"Hello world!" CENTER
INK CLUT("C64",19)
TEXT 320,200,"Hello world!" CENTER
VOICE SAY ASYNC "Hello world!"
BITMAP 1 TARGET
DO
  ROLL BITMAP LEFT 1
  PAUSE 0
LOOP WHILE INKEY$=""
```

Press F5 to run, F8 to toggle screens, ESC to exit edit mode

The Liquid BASIC Language

Variables

Variables in Liquid BASIC can be either Real (a double precision float with a range from $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$) or String (a sequence of characters that can be any length). Variable names are not case sensitive, can include letters, numbers, and the underscore '_' character, and can be any length. Append a dollar sign '$' at the end of a variable name to mark it as a String, otherwise it is assumed to be a Real.

Expressions and Operators

An expression is TRUE if it is non-zero. An expression is FALSE if it is zero.

Each binary operator (an operator with two subexpressions) has a precedence level. Operators with a higher operator precedence level are executed before operators with a lower operator precedence.

Supported operators and their precedence level include:

TOKEN	OPERATOR	PRECEDENCE
OR	Binary or	1
XOR	Binary exclusive or	1
AND	Binary and	2
=	Equal to	3
<>	Not equal to	3
<	Less than	3
>	Greater than	3
<=	Less than or equal to	3
>=	Greater than or equal to	3
SHL	Bit shift left	4
SHR	Bit shift right	4
+	Addition	5
-	Subtraction	5
MOD	Modulus	6
\	Integer division	7
*****	Multiplication	8
/	Division	8
^	Exponentiation	9

Direct Mode Commands

HELP

Print a quick reference to common commands.

NEW

Clear (erase) program and variable storage.

RUN

Execute the BASIC program in memory. Press the CTRL + C keys together to break execution.

DIR [-L] [<filter$>]

Display the contents of the current directory, filtering the results to <filter$>. If the "-L" switch is used a "long listing" is printed with the file date and file size for each file.

OPEN

Load and run a program using a File Open dialog box.

LOAD <filename$>

Load a program from disk (the .BAS file extension is optional).

SAVE <filename$>

Store the program in memory to disk.

LIST [<first>][-<last>]

List the BASIC program currently in memory.

MORE [<first>][-<last>]

List the BASIC program with page pausing.

AUTO [<inc>]

Start auto-incrementing when entering lines. Anytime a line that starts with a line number is entered the editor will automatically print the next line number incremented by <inc>. Entering a line number by itself will end auto-increment mode.

DELETE [<first>][-<last>]

Delete the range of lines from <first> to <last>. The beginning of the program is assumed if <first> is omitted. The end of the program is assumed if <last> is omitted.

RENUMBER [<start>][,<inc>]

Renumber the program in memory, starting at line number <start> and incrementing by <inc> each line. Line numbers cannot be branched to, only labels can. Line numbers are used for editing programs only.

EDIT [<filename$>]

Open the full screen editor. If an optional <filename$> is specified, the editor will load <filename$> when opened and automatically save the program back to disk when it is executed or the full screen editor is exited.

SLOW

Switch to "slow" mode. This mode locks the amount of time each frame spent on running code to provide a consistent pace and stable frame rate. This is the default mode.

FAST

Switch to "fast" mode. This mode uses all available time each frame to run code. This results in faster program execution but can have a considerable effect on the frame rate.

RESET

Perform a soft reset.

HARD RESET

Perform a hard reset.

QUIT

Exit Liquid BASIC.

General Commands

label:

An identifier followed by a colon ‘:’ is treated as a label and can be branched to.

REM

Use to comment your code. Anything on the line after REM is ignored. Comments do not affect the execution speed of your program. The apostrophe ‘’’ can be substituted for the REM keyword.

ASSERT <expr>

Use to debug your code. ASSERT is ignored if <expr> evaluates to true, otherwise program execution will stop with an “Assertion failed” error.

TRACE ON|OFF

Use to debug your code. When tracing is turned on, Liquid BASIC will print the line number of each line as it runs. This is useful to track or “trace” your code’s execution.

CLR

Clear all variables and arrays.

CLR CONST

Clear all constants.

CONST <const>=<value>[,<const>=<value>,…]

Assign a constant to a specific value. Once set the constant cannot be changed. The constant type and value type must match (e.g., Strings can only be assigned to String constants ending in a dollar sign ‘$’).

[LET] <var>=<expr>

Assign a variable to a specific value. The variable type and expression type must match (e.g., Strings can only be assigned to String variables ending in a dollar sign ‘$’). The LET keyword is optional.

GOTO <label>

Jump to <label>. It is bad programming practice to jump into or out of control blocks using the GOTO statement as it may lead to unexpected results. You cannot jump into or out of FUNCTIONs.

GOSUB <label>

Jump to <label> and save the return address (the statement after the GOSUB statement) to be later RETURN'ed to. You cannot jump into or out of FUNCTIONs.

RETURN

Return to the statement after the last GOSUB statement.

ON <expr> GOTO|GOSUB <label>[,<label>…]

Jump to the <label> in the list that <expr> indexes. For example, if <expr> evaluates to 3, then jump to the third label in the list. If no label is specified for an index no jump is performed. When GOSUB is used save the return address (the statement after the ON … GOSUB statement) to be later RETURN'ed to.

IF <expr> THEN <statements>

If <expr> evaluates to true, then run the statements after THEN.

IF <expr> THEN … [ELSE IF <expr> …][ELSE …] END IF

If <expr> evaluates to true, then run all the statements (even across multiple lines) after THEN. If <expr> evaluates to false and the optional ELSE clause is used, then run all the statements (even across multiple lines) after ELSE. END IF is required to signal the end of the IF/THEN block.

SELECT <expr> CASE <case> … [CASE <case> …][CASE ELSE …] END SELECT

Evaluate <expr> and compare to each <case> expression. If there is a match, then run all the statements (even across multiple lines) after CASE. If there is no match, then run all the statements (even across multiple lines) after CASE ELSE. END SELECT is required to signal the end of the SELECT/CASE block.

WHILE <expr> … WEND

Declare a loop. The loop will run while <expr> evaluates to true.

EXIT WHILE

Exit the current WHILE/WEND block.

DO [UNTIL|WHILE <expr>] … LOOP [UNTIL|WHILE <expr>]

Declare a loop. If the UNTIL and WHILE clauses are not used the loop will run indefinitely. DO UNTIL will run the loop until <expr> evaluates to true. DO WHILE will run the loop while <expr> evaluates to true. LOOP UNTIL repeats the loop until <expr> evaluates to true. LOOP WHILE repeats the loop while <expr> evaluates to true.

EXIT DO|LOOP

Exit the current DO/LOOP block. Either the DO or LOOP keyword can be used.

FOR <var>=<start> TO <end> [STEP <step>] … NEXT <var>

Assign <var> to <start>. If <var> is less than <end> then run the statements (even across multiple lines) before NEXT. If <var> is more than <end> at the beginning of the loop none of the loop is run. Once NEXT is encountered, increment <var> by <step> (<step> is 1 if not specified) and repeat the loop. <step> can also be negative to decrement <start> down to <end>. The <var> after NEXT is required.

EXIT FOR

Exit the current FOR/NEXT block.

RESTORE [<label>]

Set the label where the next READ command should start looking for DATA. If no <label> is specified, then READ will look for DATA starting at the beginning of the program. The <label> cannot be inside a FUNCTION.

DATA <value>[,<value>…]

Define data to be read. <value> can be either a Real or String value. DATA statements are ignored when a running program encounters them. Use the READ command to read values from a DATA statement. The data cannot be inside a FUNCTION.

READ <var>[,<var>…]

Read the next item in a DATA statement as a string and assign it to <var>. If <var> is a Real then automatically try to convert the string to Real before assigning.

BEEP

Play the host OS's default "beep" sound.

PAUSE <ms>

Pause program execution for <ms> milliseconds. PAUSE 0 immediately yields the program to the host OS and can be used to yield execution until the next vsync (short for "vertical synchronization").

RANDOMIZE <seed>

Set the seed of the random number generator to <seed>. The seed generates reproducible sequences. The same random numbers will always be generated by the RND and RANGE functions from the same seed. It is common to set the seed to TIMER as it will "randomize" (depending on the time of day) the seed and ensure decent random number generation.

CHAIN [CLR] <filename$> [RESTORE]

Load a new program into memory and run it. If the optional CLR modifier is used, then all variables and arrays are cleared before chaining. Otherwise, variables and arrays are preserved, allowing the current program to pass information to the chained program. If the optional RESTORE modifier is used the current program reloads and restarts (with all variables and arrays cleared) when the chained program ends. This is useful for menu-type programs (see Menu.bas).

STOP

Immediately stop program execution. This is the same as pressing CTRL + C together while a program is running.

END

Close all files and databases and end program execution.

Basic Input/Output

CLS

Clear the text screen.

LOCATE <x>,<y>

Move the cursor to the <x>,<y> position.

INPUT [<prompt$>;|,]var[,var…]

Input a line and assign it to <var>. If <var> is a Real then automatically try to convert the line to Real before assigning. If this fails a “Redo From Start” warning is printed and the user must re-enter their input. Multiple values may be input by separating the input with commas. Each piece of input can be wrapped in quotes to preserve any commas in it. If more items are input than expected an “Extra Ignored” warning is printed and the extra items are discarded.

If <prompt$> is omitted print a question mark ‘?’ and a space by default. Otherwise, print the <prompt$> before input. If the semicolon ‘;’ separator is used after <prompt$>, print a question mark ‘?’ and a space following the prompt. Otherwise if the comma ‘,’ separator is used, do not print “? ” after the <prompt$>.

This command cannot be used in direct mode.

LINE INPUT var$

Input a line and assign it to <var$>. <var$> must be a String variable. No parsing is done on the input; the entire line is assigned to <var$>.

This command cannot be used in direct mode.

PRINT <value>[;|,]…

Print <value> followed by a carriage return (if the optional semicolon ‘;’ is omitted) or a tab (if the optional ‘,’ is used). <value> can be a Real or a String. Multiple values can be printed. A question mark ‘?’ can be substituted for the PRINT keyword.

PRINT SPC(<x>)

Print <x> number of spaces before printing.

PRINT TAB(<x>)

Move the cursor to the <x> position before printing.

PRINT CENTER <value>[,highlight]

Centers <value> on the current line followed by a carriage return (unless on the last row of the screen, in which case the cursor returns to the start of the line). <value> can be a Real or a String. If the optional <highlight> color is specified, then the entire line will have its highlight set to it. This command can be useful for printing headers and footers.

PRINT WORDWRAP <value>[,<columns>][;]

Print <value> with automatic word wrapping followed by a carriage return (if the optional semicolon ‘;’ is omitted). The optional <columns> parameter is used to specify what column to wrap on. <value> can be a Real or a String.

WAITKEY <var$>

Pause program execution until a key is pressed and store it in var$.

Files

OPEN <filename$> FOR <access> AS [#]<handle>

Open a file using the specified file <access>:

INPUT	Used for sequential text input
OUTPUT	Used for sequential text output
APPEND	Used for sequential text output, auto-seek to the end of the file when opening
BINARY	Used for random access binary input/output

<handle> is the handle to assign this file. <filename$> is a valid filename. For INPUT file access, if the file does not exist an error is raised. For all other file accesses, if the file does not exist it will be created. Up to 256 (numbered 1 to 256) files can be opened at once.

SEEK [#]<handle>,<offset>[,<origin>]

Seek within the file assigned to <handle>. <offset> is the number of bytes to jump and can be negative if jumping backwards in the file. The optional <origin> specifies where the jump should occur from:

< 0	Beginning of file (default)
0	Current position in file
> 0	End of file

This can only be used on BINARY files.

GET #<handle>,<var>[,<var>…]

Read a single byte from the file assigned to <handle> and assign it to <var>. Multiple bytes can be read. This can only be used on BINARY files.

PUT #<handle>,<byte>[,<byte>…]

Write a single byte to the file assigned to <handle>. Multiple bytes can be written. This can only be used on BINARY files.

READ #<handle>,<var>[,<var>…]

Read a variable from the file assigned to <handle> and assign it to <var>. Multiple values can be read. This can only be used on BINARY files.

WRITE #<handle>,<value>[,<value>…]

Write a variable to the file assigned to <handle>. Real variables are stored using eight bytes and String variables are stored using one byte per character plus four bytes for the String's length. Multiple values can be written. This can only be used on BINARY files.

INPUT #<handle>,<var>[,<var>…]

Input a string (up to and including the next carriage return in the file) from the file assigned to <handle> and assign it to <var>. If <var> is a Real then automatically try to convert the string to Real before assigning. Multiple values can be inputted. This can only be used on INPUT files.

PRINT #<handle>,<value>[,|;<value>…]

Print the string <value> to the file assigned to <handle>. <value> can be a Real or a String. Multiple values can be printed. If a comma ',' delimiter is used, each value is printed on its own line. If a semicolon ';' delimiter is used, each value is printed on the same line. This can only be used on OUTPUT and APPEND files.

CLOSE [#]<handle>

Close the file assigned to <handle>.

FILE MOVE <source$>,<dest$>

Move the file <source$> to <dest$>. If <dest$> is in the same directory as <source$>, the file is simply renamed.

FILE COPY <source$>,<dest$>

Copy the file <source$> to <dest$>.

FILE DELETE <filename$>

Delete the file <filename$>.

CHDIR <path$>

Change the current directory to <path$>.

MKDIR <path$>

Create a new directory <path$>.

RMDIR <path$>

Remove the directory <path$>. The directory must be empty before it can be removed.

Arrays

DIM <var>(<dimension>)

Initialize a single dimension array. Arrays are collection of variables that can be referenced by their index. Indexes start at 0. <dimension> is the number of elements the array should have. Arrays must have at least one element and no more than one million elements. All elements are set to zero/null string when initialized.

DIM <var>(<dimension1>,<dimension2>[,<dimension3>])

Initialize a multidimensional array. <dimension1> and <dimension2> specify the number of elements a two-dimensional (matrix or grid) array should have. The optional <dimension3> specifies the number of elements a three-dimensional (cube) array should have. Arrays must have at least one element and no more than one million elements total. All elements are set to zero/null string when initialized.

REDIM <var>(<dimension1>[,<dimension2>[,<dimension3>]])

Redeclare an array. All elements are set to zero/null string when initialized.

REDIM PRESERVE <var>(<dimension1>)

Redeclare an array but save the contents of the existing array. New elements are set to zero/null string when initialized. REDIM PRESERVE can only be used on single dimension arrays.

ARRAY FILL <var>()[,<expr>]

Fill the array with <expr>. If <expr> is omitted a 0 (for Real arrays) or “” (for String arrays) is used to fill the array.

ARRAY DELETE <var>(<index>)

Delete the item at <index> in array <var>. All elements above <index> are shifted down, and a 0 (for Real arrays) or “” (for String arrays) will fill the top element. ARRAY DELETE can only be used on single dimension arrays.

ARRAY INSERT <var>(<index>)[,<expr>]

Insert the item <expr> into array <var> at <index>. All elements at <index> and above are shifted up and the top element is discarded. If <expr> is omitted a 0 (for Real arrays) or "" (for String arrays) is inserted. ARRAY INSERT can only be used on single dimension arrays.

ARRAY REVERSE <var>()[,<first>,<last>]

Reverse the elements in array <var>. If the optional <first> and <last> arguments are used, then reverse only the elements between <first> and <last>. ARRAY REVERSE can only be used on single dimension arrays.

ARRAY SHUFFLE <var>()[,<first>,<last>]

Shuffle the elements in array <var>. If the optional <first> and <last> arguments are used, then shuffle only the elements between <first> and <last>. ARRAY SHUFFLE can only be used on single dimension arrays.

ARRAY SORT <var>()[,<first>,<last>]

Sort the elements in array <var>. If the optional <first> and <last> arguments are used, then sort only the elements between <first> and <last>. ARRAY SORT can only be used on single dimension arrays.

UNDIM <var>()

Uninitialize an array, allowing the variable to be reused.

Error Handling

TRAP OFF

Disable the error handler. When a non-fatal error occurs halt program execution and print the error message and line number where the error occurred. This is the default behavior when a program is first run.

TRAP <label>

Enable the error handler at <label>. When a non-fatal error occurs record the error code and line number where the error occurred, disable further error handling, and jump to <label>. Use the reserved variables ERR and ERL to read the error code and line number where the error occurred, respectively. TRAP cannot be used inside a FUNCTION and the <label> cannot be inside a FUNCTION.

TRAP RESUME NEXT

Enable the error handler. When a non-fatal error occurs record the error code and line number where the error occurred, and resume program execution at the statement following the one that caused the error. Use the reserved variables ERR and ERL to read the error code and line number where the error occurred, respectively. TRAP RESUME NEXT cannot be used inside a FUNCTION.

RESUME <label>

When a non-fatal error occurs and is handled using the TRAP statement, re-enable error handling and resume program execution at <label>. RESUME cannot be used inside a FUNCTION and the <label> cannot be inside a FUNCTION.

RESUME NEXT

When a non-fatal error occurs and is handled using the TRAP statement, re-enable error handling and resume program execution at the statement following the one that caused the error. RESUME NEXT cannot be used inside a FUNCTION.

Functions

Functions make it easy to develop software as reusable “modules”.

Functions

FUNCTION <name>([arg[,arg…]]) … END FUNCTION

Define a function. Functions are named subroutines (blocks of code) that can be CALL’ed from anywhere. When a function has ended it returns to the statement after the CALL statement. When a function is encountered that hasn’t been CALL’ed it is simply skipped. Functions cannot be nested within one another.

Arguments can be passed to the function. Each argument is treated as a local variable within the function and freed when the function ends. Any new variables declared for the first time in the function are also treated as local variables.

Functions can optionally return a Real (if <name> does not end in a dollar sign ‘$’) or String (if <name> ends in a dollar sign ‘$’) value by assigning the value to return to the function’s <name>, as if it were a variable.

[CALL] <name>() [TO <var>]

Call the function named <name> and save the return address (the statement after the CALL statement) to be later returned to when the function has finished executing. If the optional TO modifier is used, the function’s return value is assigned to the variable <var>. Otherwise the function’s return value is discarded. The CALL keyword is optional.

EXIT FUNCTION

Exit the current FUNCTION.

Databases

Liquid BASIC uses the SQLite public domain library to access databases.

Databases

DB OPEN <filename$> AS [#]<handle>

Open a SQLite database. <handle> is the handle to assign this database. <filename$> is a valid filename to a SQLite database. If the file does not exist, it will be created. Up to 8 (numbered 1 to 8) databases can be opened at once.

DB SCALAR [#]<handle>,<sql$> [TO <var$>]

Run a scalar SQL command (that returns a single value) on the database assigned to <handle>. <sql$> is a valid SQL statement. If the optional TO modifier is used, the results are assigned to the variable <var$>. Otherwise, the results are discarded.

DB QUERY [#]<handle>,<sql$>

Query the database assigned to <handle>. <sql$> is a valid SQL statement (e.g. SELECT). The results are buffered and can be read with the DB READ command.

DB READ [#]<handle>,<var$>

Read the results from a buffered query from the database assigned to <handle> and assign it to <var$>. Use the EOQ function to determine when all the results of a query have been read.

DB WRITE [#]<handle>,<sql$> [TO <var>]

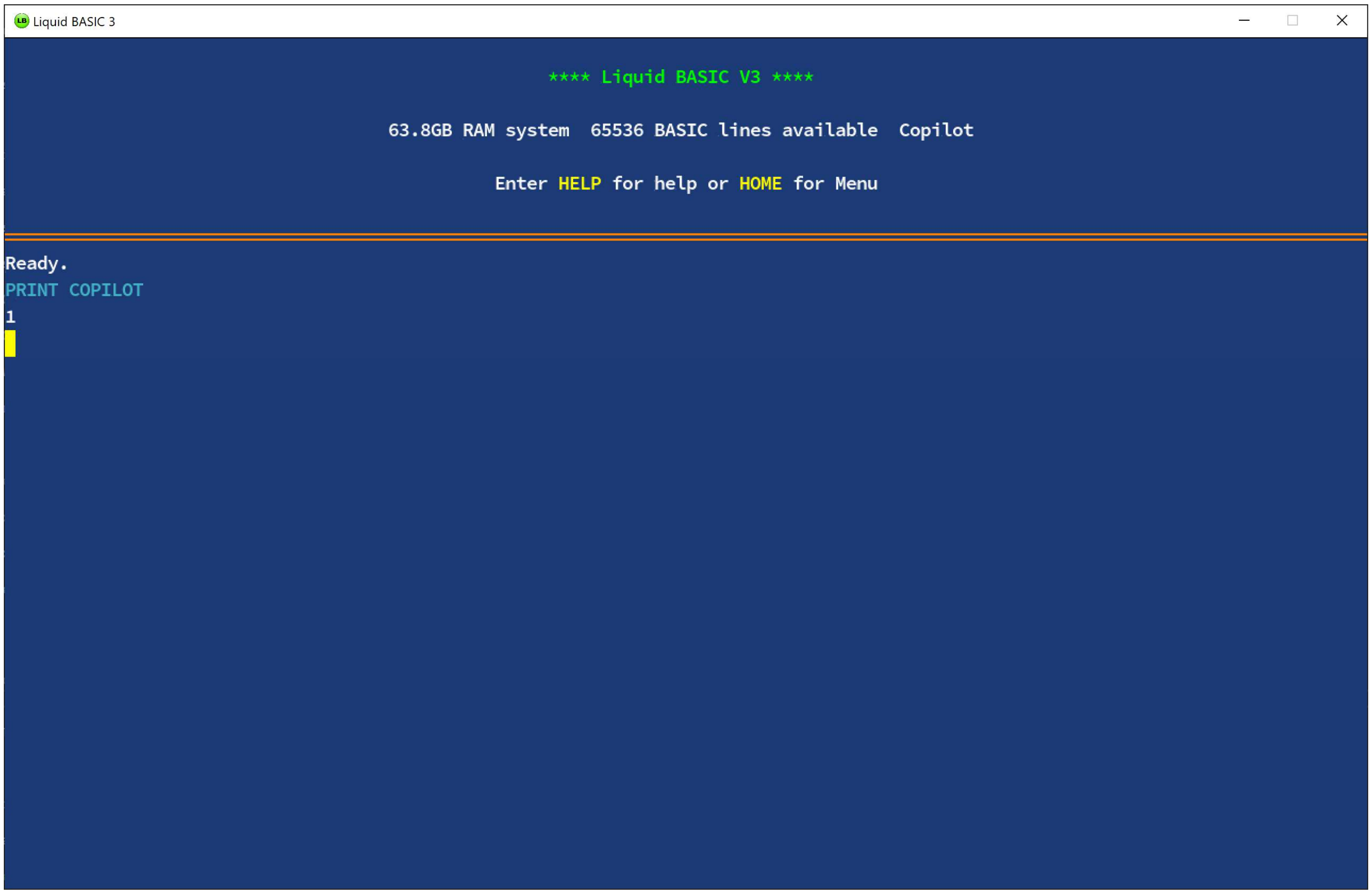
Write (execute) a SQL statement on the database assigned to <handle>. <sql$> is a valid SQL statement (e.g. UPDATE, INSERT, or DELETE). If the optional TO modifier is used, the number of rows affected are assigned to the variable <var>. Otherwise, the results are discarded.

DB CLOSE [#]<handle>

Close the database assigned to <handle>.

Artificial Intelligence

Liquid BASIC offers OpenAI integration to the ChatGPT 3.5 and DALL-E APIs. An OpenAI API key is required to make successful API calls. Sign up with OpenAI at openai.com to create your API key. Once you have your API key, paste it into a text file named "OpenAI API Key.txt" in the same directory as the Liquid BASIC executable. The next time Liquid BASIC starts up it will automatically load the OpenAI API key and validate it (it will print the word "COPILOT" next to the memory and BASIC lines available if successful). Use the COPILOT reserved variable to programmatically test whether the API key was successfully loaded:



```
**** Liquid BASIC V3 ****

63.8GB RAM system  65536 BASIC lines available  Copilot

Enter HELP for help or HOME for Menu

Ready.
PRINT COPILOT
1
```

See https://help.openai.com/en/collections/3675940-getting-started-with-openai-api for more information.

Chat

CHAT <prompt$>[,<tokens>] [TO <var$>]

Submit a one-off <prompt$> to the OpenAI ChatGPT API. <tokens> is the maximum number of tokens to return (the default is 100). You can think of tokens as pieces of words, where 1,000 tokens is about 750 words. If the optional TO modifier is used, the results are assigned to the variable <var$>. Otherwise, the results are printed to the screen.

CHATBOT NEW

Start a new conversation with the chatbot.

CHATBOT INSTRUCT <instructions$>

Submit <instruction$> to the chatbot using the System role. Instructions help set the behavior of the chatbot.

CHATBOT INPUT <prompt$>

Submit <prompt$> to the chatbot using the User role. Input is used to request a response from the chatbot. It can be provided by the end user or programmatically generated by the programmer.

CHATBOT EXAMPLE OUTPUT <output$>

Submit <prompt$> to the chatbot using the Assistant role. Example output is used to give examples of desired behavior.

CHATBOT RESPONSE TO <var$>

Call the ChatGPT API to get a response and wait until the response is delivered. The response is assigned to <var$>.

Code Generation

CODE <prompt$>[,tokens>]

Submit <prompt$> to the OpenAI Chat GPT API to write a BASIC program for you. <tokens> is the maximum number of tokens to return (the default is 1000). The program currently in memory is replaced with the results. Results are placed into program memory if the line returned starts with a line number.

Although ChatGPT is instructed to follow certain rules pertaining to Liquid BASIC, some editing or revisions may be required for the generated code to run properly.

This command can only be used in direct mode.

Text

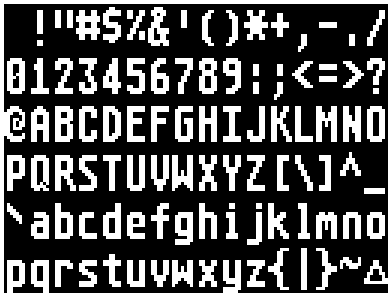
The text screen in Liquid BASIC is always at the top of the graphics stack (see "Graphics & Sound" below). By default, it is a simple 128x32 grid, with each "cell" able to hold a single character of text. Individual cells can have an independent foreground (ink) and background (highlight) color. Each cell also has a set of text attributes: blink, bold, and underline. These attributes can be turned on and off and used in any combination.

Screen Commands

SCREEN <mode>

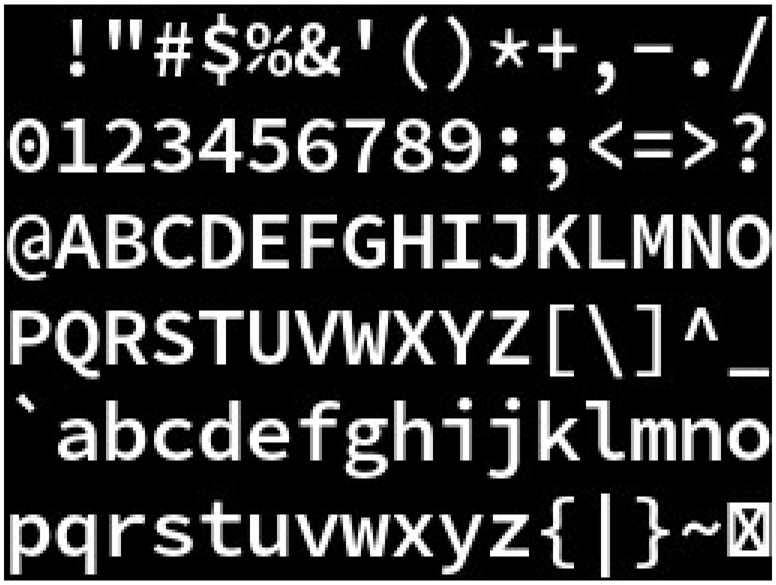
Set the text screen <mode>:

0	80x25 text, low resolution (8x16) Retro character set



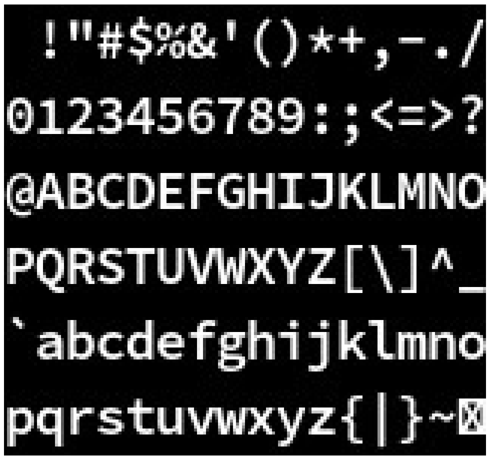
```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~△
```

1	80x25 text, high resolution (16x32) character set



```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~☒
```

2 128x32 text, high resolution (10x25) character set (default)



```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~☒
```

Color Commands

BORDER <color>

Set the border <color>.

BACKGROUND <color>

Set the background <color>.

INK <color>

Set the ink <color>.

HIGHLIGHT <color>

Set the highlight <color>.

Attribute Commands

BLINK <state>

Turn blinking on (<state> 1) or off (<state> 0).

BOLD <state>

Turn bold on (<state> 1) or off (<state> 0).

UNDERLINE <state>

Turn underline on (<state> 1) or off (<state> 0).

Graphics & Sound

Graphics in Liquid BASIC are comprised of Layers and Sprites. There are 8 Layers, numbered from 1 to 8. Each Layer is "stacked" on top of one another (with Layer 1 at the bottom and Layer 8 at the top). There are up to 256 Sprites, numbered from 1 to 256. Sprites have a definable priority to determine the order they are rendered and can be placed in front of any Layer.

Both Layers and Sprites can be TileMap-based or Bitmap-based. A TileMap is similar to a text screen, comprised of multiple "cells" in a grid that are made up of tiles from a TileSet. Bitmaps are comprised of individual pixels (short for "picture elements") in a grid, each of which can be set to its own individual color.

Layers can be any one of six predefined modes or use a user defined TileMap or Bitmap to create a custom mode. They have a fixed position, and can be independently shown, hidden, scaled, rotated, and tinted. They can also be "fit" to the screen (center, stretch, fit, fill, or tile). And they can be mirrored horizontally or flipped vertically.

Sprites are 2D moveable objects that use a TileMap or Bitmap to render. They can be shown, hidden, translated (moved), scaled, rotated, and tinted. And they can be mirrored horizontally or flipped vertically. Sprites also have simple collision detection to determine when two or more Sprites touch or intersect each other.

Using Layers and Sprites it is possible to create impressive demos, games, and other graphical effects. TileMaps make it easy to scroll the entire screen in any direction. Multiple Layers can be used for parallax scrolling, transparency, and other special effects. Sprites can be used for individual characters, NPC's, and other moving objects.

About Colors

Colors in Liquid BASIC are represented by a 32-bit unsigned integer, comprised of four 8-bit bytes. Each color has a byte for the red, green, blue, and alpha (transparency) levels of the color. Together this creates 16,777,216 color combinations with 256 levels of transparency.

Individual pixels are essentially colors. The alpha level is used to determine transparency; it determines how to blend a color with the pixel beneath it when it is being written (0 is completely transparent, 255 is completely opaque, and any value in-between is a level of transparency).

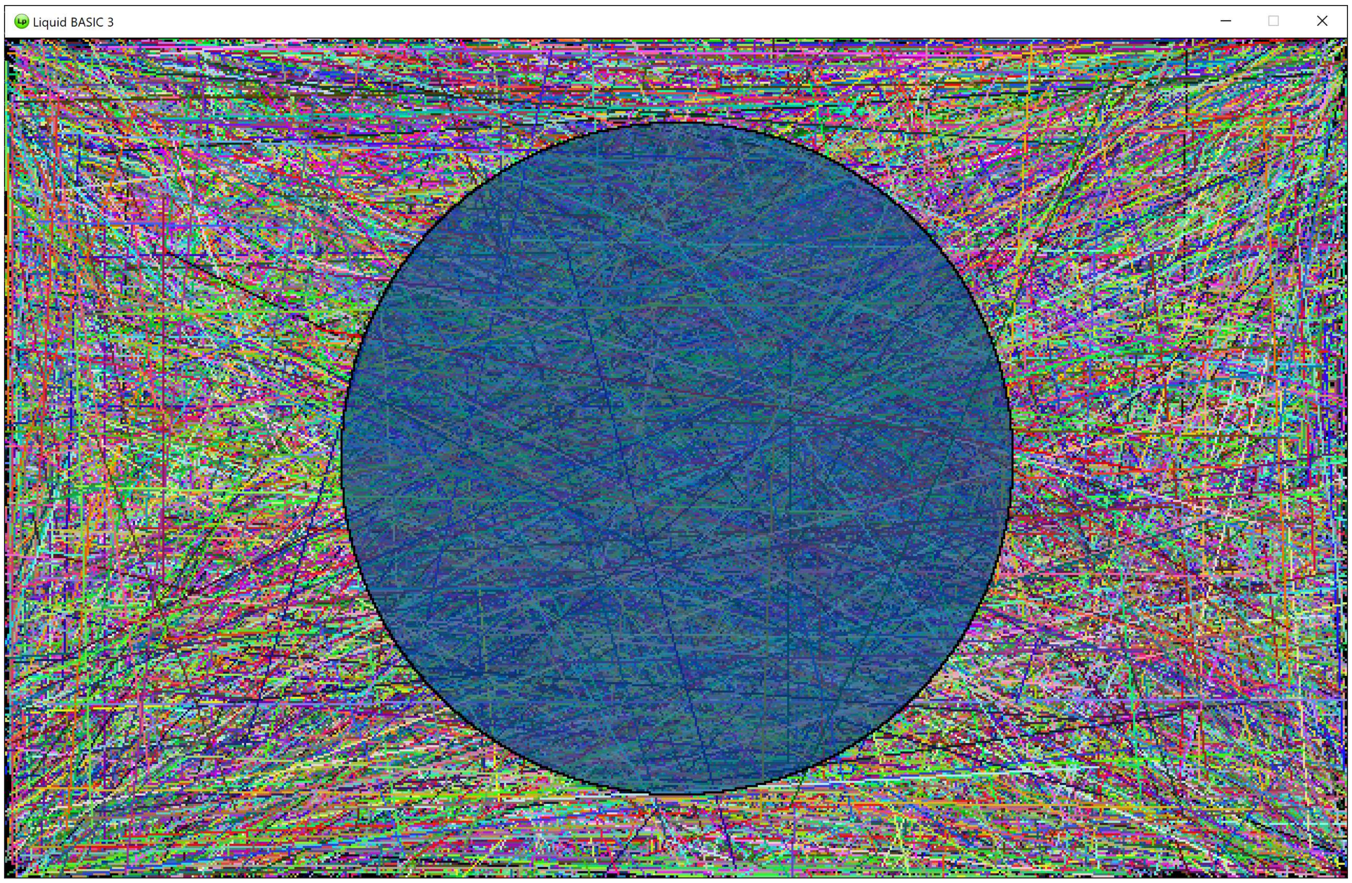
The easiest way to get a proper color value is to use the built-in HSV and RGB functions.

About Bitmaps

Bitmaps are 2D rectangular grids of pixels. They can be any size from 1x1 pixel up to the graphics card limit (typically 4096x4096 pixels on modern PCs). Each pixel is a color value and can be directly read from and written to. Anytime a pixel is written to it is first clipped to the current clipping region (pixels outside of the clipping region are not written) and the current pixel operator and pixel mask are applied (unless otherwise noted).

The pixel operator takes the “source” pixel (the pixel being written) and the “destination” pixel (the pixel presently in the Bitmap) and applies an operator to it. The simplest (and default) operator is the write operator: the source pixel simply overwrites the destination pixel. The blend operator is also useful: the source pixel is blended with the destination pixel to allow transparency effects. Other operators are used to fade pixels in or out, or average two pixels together.

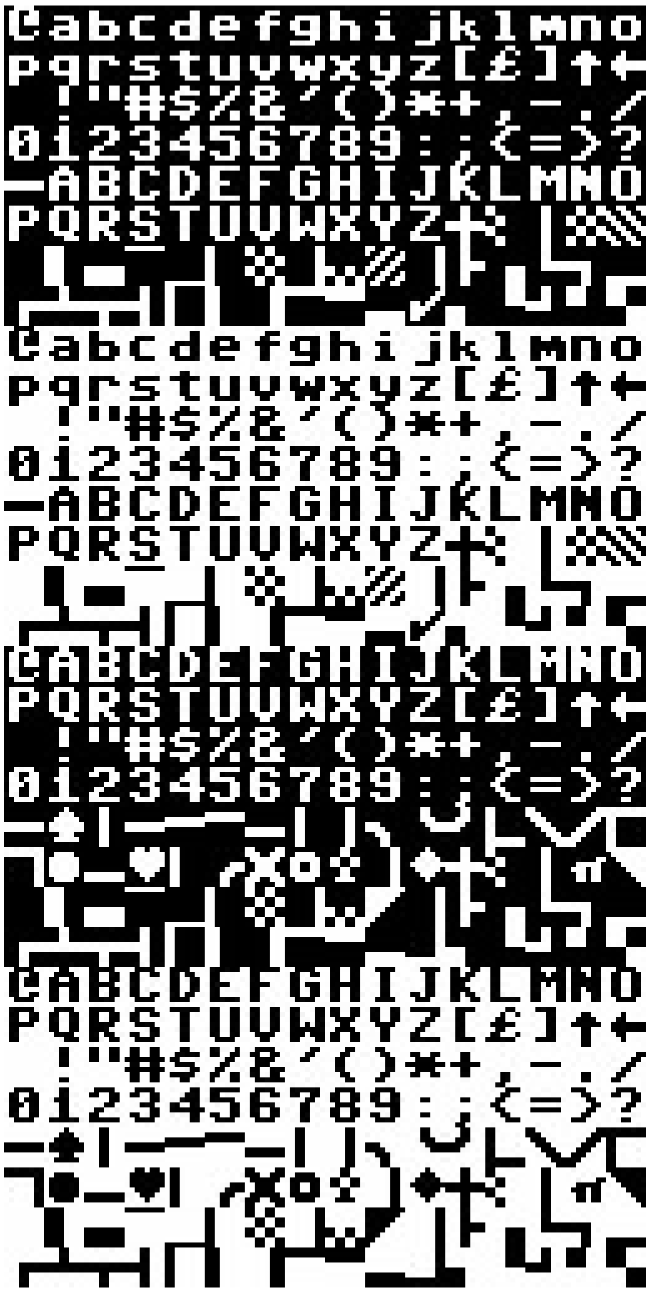
The pixel mask is a 32-bit mask to apply to each pixel written. It defines which bits in a 32-bit color can pass through and be written. By default, all 32-bits are enabled, so all bits are written. With pixel masks it is possible to restrict pixel writes to the red, green, blue, or alpha bytes that comprise a color, or some combination of bits thereof.



About Tiles, TileSets, and TileMaps

Tiles are small (usually 8x8 pixels or 16x16 pixels) pieces of a larger Bitmap that are collected into a TileSet. A TileSet can contain any number of tiles, limited only by the size of the Bitmap that it is built on. Individual tiles can be cleared, customized, and scrolled.

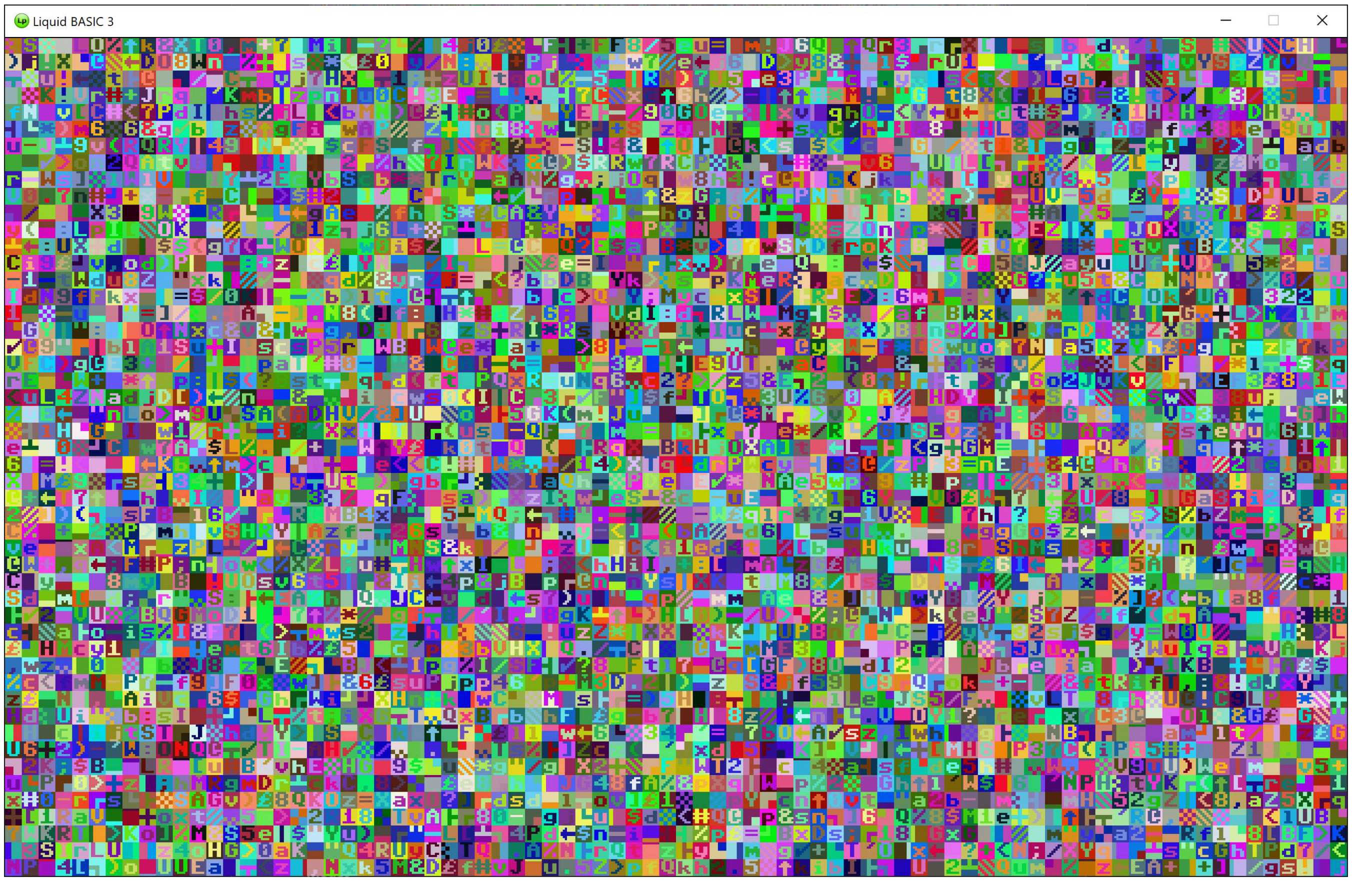
TileSet 1 is reserved for the default (Commodore 64-based) TileSet. This read-only TileSet has 512 8x8 pixel tiles, with a combination of uppercase, lowercase, and graphics characters (known as PETSCII):



For the default TileSet, tiles 0 to 95 are mapped to their respective ASCII characters:

```
@abcdefghijklmno
pqrstuvwxyz[ ]^
 !"#$%&'()*+,-./
0123456789:;<=>?
 ABCDEFGHIJKLMNO
PQRSTUVWXYZ
```

A TileMap is a 2D rectangular grid of “cells”. They can be any size from 1x1 cell up to 256x256 cells. Each cell can hold a tile, a foreground (ink) color, and a background (highlight) color and can be directly read from and written to. Because TileMaps use indexes to tiles, which in turn represent patterns of pixels in a larger Bitmap, entire sections of the screen can be quickly filled, manipulated, and scrolled by simply changing the indexes. Individual tiles can also be customized, and changes to a tile are reflected anywhere that tile is used in a TileMap. This allows large portions of the screen to be animated or modified while changing a minimal amount of data. The smaller dataset that TileMaps work with generally perform faster than the same operations in Bitmaps. However, Bitmaps allow more flexibility in that every single pixel can be independently manipulated.



Modes

MODE <mode>[,<id>]

The MODE command is used to set the current Layer to one of six pre-defined modes:

0	None
1	40x25 TileMap
2	80x50 TileMap
3	160x100 TileMap
4	320x200 Bitmap
5	640x400 Bitmap
6	1280x800 Bitmap

The next valid free TileMap or Bitmap is used if the optional <id> is omitted. Otherwise, the specified TileMap or Bitmap <id> is used and assigned to the current Layer.

Changing modes using the MODE command also redirects the “target” of TileMap and Bitmap commands so all subsequent commands will render to the selected mode.

Alternatively, custom modes of any size TileMap or Bitmap can be explicitly defined using the LAYER # USE TILEMAP|BITMAP commands.

Buffering Commands

SINGLE BUFFER

Make the TileMap or Bitmap single buffered. Subsequent commands draw to the front buffer, so changes are immediately visible.

DOUBLE BUFFER

Make the TileMap or Bitmap double buffered. Subsequent commands draw to the back buffer in memory, so changes are not visible. When the buffers are swapped with the SWAP BUFFERS command the back buffer becomes visible (front buffer) and the front buffer then becomes the back buffer. This is useful during animation to prevent “screen tearing” and flickering, when the TileMap or Bitmap is shown before rendering has completed.

SWAP BUFFERS

Swap the back and front buffer in a double buffered TileMap or Bitmap.

TileSet Commands

CLEAR TILE <tile>

Clear (erase) the tile <tile>.

PALETTE <ch>,<color>

Assign a color to a character. This character then acts as a substitute for the color value when defining custom characters using the CUSTOM TILE command. <ch> is the ASCII code of the character to assign <color> to.

CUSTOM TILE <tile>,<scanline>,<data$>

Set the <scanline> in tile <tile> to <data$>. <data$> should be the same length as the width of the tile, and use characters defined in PALETTE to substitute color pixels.

ROLL TILE <tile> UP|DOWN|LEFT|RIGHT <count>

Roll the tile <tile> in the direction specified by <count> pixels. When rolling, the pixels that leave one side of the tile appear on the other side.

SCROLL TILE <tile> UP|DOWN|LEFT|RIGHT <count>

Scroll the tile <tile> in the direction specified by <count> pixels. When scrolling, the pixels that leave one side of the tile disappear and blank space appears on the other side.

TileMap Commands

CLEAR TILEMAP

Clear the active targeted TileMap.

POKE TILE <index>,<tile>

Poke the tile <tile> at <index>. Use the PEEKTILE function to read the tile back.

POKE INK <index>,<ink>

Poke the foreground color <ink> at <index>. Use the PEEKINK function to read the foreground color back.

POKE HIGHLIGHT <index>,<highlight>

Poke the background color <highlight> at <index>. Use the PEEKHIGHLIGHT function to read the background color back.

PUT TILE <x>,<y>,<tile>

Draw the tile <tile> at <x>,<y> using the current ink and highlight color.

PUT TILES <x>,<y>,<tiles$>

Draw a string of tiles <tiles$> at <x>,<y> using the current ink and highlight color.

PUT HLINE <x1>,<x2>,<y>,<tile>

Draw a horizontal line from <x1>,<y> to <x2>,<y> with <tile> using the current ink and highlight color.

PUT VLINE <x>,<y1>,<y2>,<tile>

Draw a vertical line from <x>,<y1> to <x>,<y2> with <tile> using the current ink and highlight color.

PUT RECT [FILL] <x1>,<y1>,<x2>,<y2>,<tile>

Draw a rectangle from <x1>,<y1> to <x2>,<y2> with <tile> using the current ink and highlight color. Use FILL to fill the rectangle.

SHIFT TILEMAP UP|DOWN|LEFT|RIGHT <count>

Smooth scroll the TileMap at the pixel level in the specified direction <count> pixels. Use the SHIFTX and SHIFTY functions to read the shift X and shift Y registers back.

SHIFT TILEMAP <dx>,<dy>

Smooth scroll the TileMap at the pixel level to the specified offset <dx>,<dy>.

ROLL TILEMAP UP|DOWN|LEFT|RIGHT <count> [WINDOW <x1>,<y1>,<x2>,<y2>]

Roll the screen in the direction specified by <count> tiles. When rolling, the tiles that leave one side of the TileMap appear on the other side. Use the optional WINDOW modifier to roll a section of the screen and not the entire screen.

SCROLL TILEMAP UP|DOWN|LEFT|RIGHT <count> [WINDOW <x1>,<y1>,<x2>,<y2>]

Scroll the screen in the direction specified by <count> tiles. When scrolling, the tiles that leave one side of the TileMap disappear and blank space appears on the other side. Use the optional WINDOW modifier to scroll a section of the screen and not the entire screen.

Bitmap Commands

CLEAR BITMAP

Clear the active targeted Bitmap.

CLIP <x1>,<y1>,<x2>,<y2>

Set the clipping region. Pixels outside of the region are not written.

SMOOTH <state>

Smooth (apply linear filter when rendering) the Bitmap if <state> is non-zero. Otherwise render without filtering.

PIXEL OPERATOR <op>

Set the current pixel operator. The pixel operator is applied anytime a pixel is written to a Bitmap. <op> must be between 1 and 14. Several constants are included to simplify the pixel operators:

	CONSTANT	**ACTION**
1	PXO_WRITE (default)	overwrite the destination (d) pixel with the source (s) pixel
2	PXO_BLEND	blend source and destination pixels using the alpha channel
3	PXO_MIN	`d = min(s, d)`
4	PXO_MAX	`d = max(s, d)`
5	PXO_AND	`d = s & d`
6	PXO_OR	`d = s | d`
7	PXO_XOR	`d = s ^ d`
8	PXO_ADD	`d = s + d`
9	PXO_SUB	`d = s - d`
10	PXO_AVG	`d = (s + d) >> 1`
11	PXO_MULT	`d = (s * d) / 255`
12	PXO_ZERO	only write if destination pixel has no RGB data
13	PXO_REPLACE	only write if destination pixel has RGB data
14	PXO_ALPHATEST	only write if source pixel has alpha data

Pixel operators act on a pixel's individual bytes, not the entire 32-bit pixel. For example, the add pixel operator will add the red byte, the green byte, the blue byte, and the alpha byte independently. Each will be clamped to 255 so it doesn't exceed what can be stored in a single byte. Conversely, the subtract pixel operator will clamp each byte to 0 when subtracting.

PIXEL MASK <mask>

Set the current pixel mask. The pixel mask is applied anytime a pixel is written to a Bitmap. The pixel mask is referenced in ABGR format, where A is the alpha byte, B is the blue byte, G is the green byte, and R is the red byte. Each byte is 8 bits and can hold a value between 0 and 255.

Some useful <mask> values:

`$FFFFFFFF`	write the entire pixel (default)
`$000000FF`	only write the red byte
`$0000FF00`	only write the green byte
`$00FF0000`	only write the blue byte
`$FF000000`	only write the alpha byte
`$00FFFFFF`	write the red-green-blue bytes (but not the alpha byte)

LINE STIPPLE <stipple>

Define the 32-bit line stipple pattern (which pixels to draw when drawing lines). This is useful for drawing dotted and dashed lines.

Some useful <stipple> values:

`$FFFFFFFF`	solid line (default)
`$55555555`	dotted line (small dots)
`$33333333`	dotted line (big dots)
`$0F0F0F0F`	dashed line

USE PATTERN <id>

Set the Pattern to use when pattern filling graphics primitives (like STRIP, RECT, CIRCLE, ELLIPSE, and FLOOD FILL) to Pattern <id>.

USE TILESET <id>

Set the TileSet to use when drawing tiles to TileSet <id>.

USE FONT <id>

Set the Font to use when printing text to Font <id>.

POKE <index>,<color>

Poke (write directly to memory) the pixel <color> at <index>. Clipping, the pixel operator, and the pixel mask are ignored when poking a pixel. Use the PEEK function to read the pixel back.

SWAP PIXELS <oldColor>,<newColor> [WINDOW <x1>,<y1>,<x2>,<y2>]

Swap <oldcolor> with <newcolor>. Use the optional WINDOW modifier to swap a section of the screen and not the entire screen. When swapping pixels clipping, the pixel operator, and the pixel mask are ignored.

PLOT <x>,<y>[,<x>,<y>…]

Plot a pixel at <x>,<y> using the current ink color. Multiple pixels can be plotted. Use the POINT and SAMPLE functions to read the pixel back.

HLINE <x1>,<x2>,<y>

Draw a horizontal line from <x1>,<y> to <x2>,<y>.

VLINE <x>,<y1>,<y2>

Draw a vertical line from <x>,<y1> to <x>,<y2>.

LINE <x1>,<y1>,<x2>,<y2> [TO <x3>,<y3>…]

Draw a line from <x1>,<y1> to <x2>,<y2>. Multiple lines can be drawn with the TO modifier.

RADIAL LINE <x>,<y>,<dir>,<step>

Draw a radial line at <x>,<y> in the direction <dir> for <step> pixels.

SMOOTH LINE <x1>,<y1>,<x2>,<y2> [TO <x3>,<y3>…]

Draw a smooth, anti-aliased line from <x1>,<y1> to <x2>,<y2>. Multiple lines can be drawn with the TO modifier.

STRIP [PATTERN [FILL]] <x1>,<y1>,<x2>,<y2>,<x3>,<y3>

Draw a triangle from <x1>,<y1> to <x2>,<y2> to <x3>,<y3>. Use FILL to fill the triangle or PATTERN FILL to fill the triangle with the active Pattern.

RECT [PATTERN [FILL]] <x1>,<y1>,<x2>,<y2>

Draw a rectangle from <x1>,<y1> to <x2>,<y2>. Use FILL to fill the rectangle or PATTERN FILL to fill the rectangle with the active Pattern.

CIRCLE [PATTERN [FILL]] <x>,<y>,<r>

Draw a circle at <x>,<y> with the radius <r>. Use FILL to fill the circle or PATTERN FILL to fill the circle with the active Pattern.

ELLIPSE [PATTERN [FILL]] <x>,<y>,<xr>,<yr>

Draw an ellipse at <x>,<y> with the radius <xr> and <yr>. Use FILL to fill the ellipse or PATTERN FILL to fill the ellipse with the active Pattern.

BEZIER CURVE <x1>,<y1>,<x2>,<y2>,<x3>,<y3>,<x4>,<y4>

Draw a Bezier curve within (x1,y1) and (x2,y2) and (x3,y3) and (x4, y4).

FLOOD [PATTERN] FILL <x>,<y>

Flood fill the enclosed area at <x>,<y>. Use PATTERN to flood fill with the active Pattern.

TEXT <x>,<y>,<value> [ALIGN <alignX>,<alignY>]

Print the text <value> at <x>,<y> using the current Font. The optional ALIGN modifier is used to align the text around <x>,<y>:

<alignX>:	-1 (left)	0 (center)	1 (right)
<alignY>:	-1 (top)	0 (center)	1 (bottom)

The default alignment is to the upper left corner of the tile (-1, -1).

PASTE <x>,<y>,<data$>

Paste the copied <data$> (see the COPY$ function) to <x>,<y>.

ROLL BITMAP UP|DOWN|LEFT|RIGHT <count> [WINDOW <x1>,<y1>,<x2>,<y2>]

Roll the screen in the direction specified by <count> pixels. When rolling, the pixels that leave one side of the Bitmap appear on the other side. Use the optional WINDOW modifier to roll a section of the screen and not the entire screen.

SCROLL BITMAP UP|DOWN|LEFT|RIGHT <count> [WINDOW <x1>,<y1>,<x2>,<y2>]

Scroll the screen in the direction specified by <count> pixels. When scrolling, the pixels that leave one side of the Bitmap disappear and blank space appears on the other side. Use the optional WINDOW modifier to scroll a section of the screen and not the entire screen.

STAMP TILE <tile> TO [BITMAP <id>,]<x>,<y> [ALIGN <alignX>,<alignY>]

Stamp the tile <tile> to <x>,<y> using the current TileSet. The optional BITMAP modifier is used to stamp to a Bitmap that is not the current target. The optional ALIGN modifier is used to align the tile around <x>,<y>.

STAMP TILES <tile$> TO [BITMAP <id,]<x>,<y> [ALIGN <alignX>,<alignY>]

Stamp a string of tiles <tile$> to <x>,<y> using the current TileSet. The optional BITMAP modifier is used to stamp to a Bitmap that is not the current target. The optional ALIGN modifier is used to align the tiles around <x>,<y>.

STAMP TILEMAP <id> TO [BITMAP <id,]<x>,<y> [ALIGN <alignX>,<alignY>]

Stamp the TileMap <id> to <x>,<y>. The optional BITMAP modifier is used to stamp to a Bitmap that is not the current target. The optional ALIGN modifier is used to align the TileMap around <x>,<y>.

STAMP BITMAP <id> TO [BITMAP <id>,]<x>,<y> [ALIGN <alignX>,<alignY>]

Stamp the Bitmap <id> to <x>,<y>. The optional BITMAP modifier is used to stamp to a Bitmap that is not the current target. The optional ALIGN modifier is used to align the Bitmap around <x>,<y>.

STAMP BITMAP <id>,<name$> TO [BITMAP <id>,]<x>,<y> [ALIGN <alignX>,<alignY>]

Stamp the Bitmap's subtexture <name$> to <x>,<y>. The optional BITMAP modifier is used to stamp to a Bitmap that is not the current target. The optional ALIGN modifier is used to align the Bitmap's subtexture around <x>,<y>.

The Blitter

The Blitter ("bit block transfer") is used to quickly fill, copy, and manipulate data to and within a Bitmap. It can also use 3x3 matrices and matrix multiplication to apply affine transformations (such as translating, scaling, rotating, and shearing) to a Bitmap before rendering. This is useful to create "software sprites", pseudo 3D graphics, and other effects. The Blitter is software-based but optimized and is much faster than running equivalent routines written in Liquid BASIC.

The current ink color, clipping region, pixel operator, and pixel mask are all applied when the Blitter writes pixels.

Blitter Commands

BLIT FILL [BITMAP <id>,]<color> [WINDOW <x1>,<y1>,<x2>,<y2>]

Quickly fill a region of a Bitmap. The optional BITMAP modifier is used to fill a Bitmap that is not the current target. Use the optional WINDOW modifier to fill a section of the screen and not the entire screen.

BLIT COPY [BITMAP <id>,]<x1>,<y1>,<x2>,<y2> TO [BITMAP <id>,]<x>,<y>

Quickly copy a region of a Bitmap to another region. The optional BITMAP modifiers are used to copy from or to a Bitmap that is not the current target.

Blitter Affine Transformation Commands

BLIT RESET

Reset the Blitter's matrices and clears its cache.

BLIT TRANSLATE <x>,<y>

Translate (move) the current matrix <x>,<y> units.

BLIT SCALE <sx>,<sy>

Scale the current matrix by <sx>,<sy> units.

BLIT ROTATE <degrees>

Rotate the current matrix by <degrees> degrees.

BLIT SHEAR <sx>,<sy>

Shear the current matrix by <sx>,<sy>.

BLIT BITMAP <id>

Stamp the Bitmap <id> using the current matrix for affine transformations.

Blitter Frustum Commands

BLIT FRUSTUM <id>,<worldX>,<worldY>,<worldA>,<near>,<far>,<fovHalf> [WINDOW <x1>,<y1>,<x2>,<y2>]

Project a view frustum onto Bitmap <id> and render. This is useful for SNES Mode 7-style effects. <worldX>, <worldY>, and <worldA> are the coordinates of the camera. <near> is how near the camera the frustrum should start, and <far> is how far from the camera the frustrum should end. <fovHalf> is the field of view, divided in half (PI 'π' divided by 4 is a good value to use). Use the optional WINDOW modifier to render to a section of the screen and not the entire screen.

The Copper

The Copper is a special layer that sits directly above the background color. It allows each individual scanline to be set to its own color. The Copper can also show, hide, clear, fill, and scroll the scanlines.

Copper Commands

COPPER SHOW

Show the copper.

COPPER HIDE

Hide the copper (default).

COPPER CLEAR

Clear the copper (set all scanlines to 0 – transparent).

COPPER FILL <color>

Fill the copper with <color>.

COPPER SET <scanline>,<color>

Set individual scanline <scanline> (0 to 799) to the color <color>.

COPPER ROLL UP|DOWN <count>

Roll the copper colors in the direction specified by <count> scanlines. When rolling, the scanlines that leave one side of the Copper appear on the other side.

COPPER SCROLL UP|DOWN <count>

Scroll the copper colors in the direction specified by <count> scanlines. When scrolling, the scanlines that leave one side of the Copper disappear and blank space appears on the other side.

Filters

Filters are applied to the entire Bitmap and are used to create special effects and otherwise manipulate the Bitmap. Filters are software-based but optimized and are much faster than running equivalent routines written in Liquid BASIC.

Filter Commands

APPLY FILTER <command$>[,<arguments$>]

Run the <command$> filter. The <command$> is automatically converted to lowercase letters and all leading and trailing spaces removed. Optional <arguments$> can be passed in a comma separated list. Use the EXPORT$ function to help with building a comma separated list from one or more expressions.

Filter Commands and Arguments

adjusthsv <h>,<s>,<v>

Adjust the Bitmap's hue <h>, saturation <s>, and brightness <v>.

adjustrgb <r>,<g>,<b>

Adjust the Bitmap's red <r>, green <g>, and blue <b>.

blur

Blur the Bitmap (un-sharpen).

dilate

Dilate the Bitmap (bright areas grow bigger).

edgehorizontal

Find the edges of the Bitmap horizontally.

edgevertical

Find the edges of the Bitmap vertically.

emboss

Emboss the Bitmap (faked 3D look).

equalizefull

Equalize the Bitmap (adjust histogram fully).

equalizestretch

Equalize the Bitmap (adjust histogram to expose more details in the image).

erode

Erode the Bitmap (dark areas grow bigger).

grayscale

Gray scale the Bitmap.

invert

Invert the Bitmap.

kaleidoscope <corner>

Kaleidoscope the Bitmap (mirror and resize the bitmap 2x2).

logpolar

Perform a log polar effect on the Bitmap (transform the picture like the human brain does).

maketileable <strength>

Make the Bitmap tileable (prepare borders to make tile distort look better).

mediancut

Median cut the Bitmap (similar to blur).

motionblur <strength>

Motion blur the Bitmap.

move <dx>,<dy>

Move distort the Bitmap to <dx>,<dy>.

noise <seed>,<radius>

Add noise to the Bitmap (noisy and natural look).

pixels <color>,<count>

Plot a number of random pixels in the Bitmap.

scalehsv <h>,<s>,<v>

Scale the Bitmap's hue <h>, saturation <s>, and brightness <v>.

scalergb <r>,<g>,<b>

Scale the Bitmap's red <r>, green <g>, and blue <b>.

sculpture

Sculpture the Bitmap (similar to a 3D chrome effect).

sepia

Sepia the Bitmap (old-timey 1800's effect).

sharpen

Sharpen the Bitmap (expose more details in the image).

sine <dx>,<depthx>,<dy>,<depthy>

Sine distort the Bitmap.

sinergb <r>,<g>,<b>

Sine the Bitmap's red <r>, green <g>, and blue <b> colors.

tile

Tile the Bitmap (copy and resize the bitmap 2x2).

tunnel <zoom>

Tunnel the Bitmap.

twirl <rot>,<scale>

Twirl the Bitmap.

wood

Create woody/psychedelic colors in the Bitmap.

Resources

A resource is a block of memory used for a specific task. Each resource has a unique integer ID to identify it. Resource IDs start at 1 and go up to a defined limit depending on the type of resource. Available resources, their limits, and supported file formats they can be loaded from and saved to include:

RESOURCE	LIMIT	LOAD FILE FORMATS	SAVE FILE FORMATS
TileSet	64	LTS	LTS
TileMap	256	LTM	LTM
Font	16	OTF, TTF	-
Bitmap	1024	BMP, GIF, ICO, IFF, JPG, LBM, PNG, TGA, WEBP, XML (Sprite sheets)	BMP, GIF, JPG, PNG, TGA
Pattern	64 (32 available to the user)	Same as Bitmap (except Sprite Sheets)	Same as Bitmap
Layer	8	-	-
Sprite	256	-	-
Sound	256	WAV, MP3, CDA, AAC, ADT, ADTS, FLAC	-

Liquid BASIC automatically handles allocating and deallocating resources as they are used, freeing the programmer from having to.

To use a resource, it first must be created or loaded. To edit a resource, it must be targeted so subsequent commands are directed at it. When a resource is targeted, it becomes the “active” resource. For example, if Sprite 3 is targeted, then the active Sprite is said to be 3.

Resources allow memory to be abstracted, allowing almost unlimited approaches to graphics and sound. Multiple TileSets which use the same tiles but have animations between them can be quickly swapped into a TileMap, updating all the tiles within it with the new tile data and providing “cheap” (performant) animation. Bitmaps can be targeted and rendered off-screen, allowing for procedurally generated artwork. Moreover, both TileMaps and Bitmaps can be used for Layers and Sprites, offering a lot of flexibility in how to use these resources. The possibilities are endless.

Clear Command

CLEAR ALL

Free all resources and set the active resource back to 1. It is good practice to start your program with this command to ensure a clean slate at startup.

CLEAR TILESETS

Free all TileSets and set the active TileSet to 1.

CLEAR TILEMAPS

Free all TileMaps and set the active TileMap to 1.

CLEAR FONTS

Free all Fonts and set the active Font to 1.

CLEAR BITMAPS

Free all Bitmaps and set the active Bitmap to 1.

CLEAR PATTERNS

Free all Patterns and set the active Pattern to 1.

CLEAR LAYERS

Free all Layers and set the active Layer to 1.

CLEAR SPRITES

Free all Sprites and set the active Sprite to 1.

CLEAR SOUNDS

Free all Sounds and set the active Sound to 1.

Chaining Resource Modifiers

When using resources, modifiers can be “chained” together on a single line for convenience. For example:

SPRITE 1 SHOW MOVE 320,200 SCALE 2,2

is the same as:

SPRITE 1 SHOW
SPRITE 1 MOVE 320,200
SPRITE 1 SCALE 2,2

TileSets

TileSet Modifiers

TILESET <id> NEW <width>,<height>

Create a new, blank TileSet with 256 tiles. Each tile is <width> by <height> pixels large. No characters are mapped.

TILESET <id> NEW <width>,<height> USE BITMAP <bitmapId>

Create a new TileSet using the Bitmap <bitmapId> for pixel data. Each tile is <width> by <height> pixels large. The number of tiles created depends on how many tiles can fit in the size of the Bitmap. For example, a 512x256 Bitmap with 8x8 tiles will have 2048 tiles:

- 512/8 = 16, 256/8 = 8, 16x8 = 2048 tiles

No characters are mapped.

TILESET <id> COPY TILESET <tileSetId>

Copy the TileSet <tileSetId> and assign it to the unique identifier <id>.

TILESET <id> LOAD <filename$>

Load a TileSet resource and assign it to the unique identifier <id>. <filename$> is the valid filename of a file in a supported TileSet file format.

TILESET <id> SAVE <filename$>

Save the TileSet <id> to disk in a supported TileSet file format.

TILESET <id> TARGET

Target the TileSet <id> (must be between 1 and 64). All subsequent TileSet commands are directed at the targeted TileSet.

TILESET <id> MAP <ch>[,count] TO <tile>

Map the character <ch> (and the number of <count> characters after) to <tile>.

TILESET <id> MAP <ch$> TO <tile>

Map the characters in <ch$> to the tiles starting at <tile>.

TILESET <id> MAP ALL

Map all available characters starting at tile 0.

TILESET <id> FREE

Free a TileSet from memory.

TileMaps

TileMap Modifiers

TILEMAP <id> NEW <columns>,<rows>

Create a new TileMap using TileSet 1 (the default Commodore 64-based TileSet). <columns> is the number of cells horizontally, <rows> is the number of cells vertically.

TILEMAP <id> NEW <columns>,<rows> USE TILESET <tilesetId>

Create a new TileMap using the specified TileSet <tileSetId>. <columns> is the number of cells horizontally, <rows> is the number of cells vertically.

TILEMAP <id> COPY TILEMAP <tileMapId>

Copy the TileMap <tileMapId> and assign it to the unique identifier <id>.

TILEMAP <id> LOAD <filename$>

Load a TileMap resource and assign it to the unique identifier <id>. <filename$> is the valid filename of a file in a supported TileMap file format.

TILEMAP <id> SAVE <filename$>

Save the TileMap <id> to disk in a supported TileMap file format.

TILEMAP <id> TARGET

Target the TileMap <id> (must be between 1 and 256). All subsequent TileMap commands are directed at the targeted TileMap.

TILEMAP <id> FREE

Free a TileMap from memory.

Fonts

Fonts are used when rendering text in a Bitmap. Liquid BASIC supports both TTF (TrueType Font) and OTF (OpenType Font) formats.

Font Modifiers

FONT <id> LOAD <filename$>,<size>

Load a Font from disk. <id> must be between 1 and 16. <filename$> is the valid filename of a file in a supported Font file format. <size> is the point size of the Font. Point size refers to the height of the font. In digital publishing, 1 point = 1/72 of an inch.

FONT <id> FREE

Free a Font from memory.

Bitmaps

Bitmap Modifiers

BITMAP <id> NEW <width>,<height>

Create a new Bitmap that is <width> pixels across and <height> pixels down and assign it to <id>.

BITMAP <id> NEW TEXTURE <command$>[,<arguments$>]

Create a new 256x256 Bitmap using the <command$> texture generator (see below) and assign it to Bitmap <id>. The <command$> is automatically converted to lowercase letters and all leading and trailing spaces removed. Optional <arguments$> can be passed in a comma separated list. Use the EXPORT$ function to help with building a comma separated list from one or more expressions.

BITMAP <id> NEW IMAGE <prompt$>[,<size>]

Submit <prompt$> to the OpenAI DALL-E API. <size> is the size of the image and must be 256, 512, or 1024 (the default is 256). Assign the results to the bitmap <id>. This command requires an OpenAI API Key.

BITMAP <id> COPY BITMAP <bitmapId>

Copy the Bitmap <bitmapId> and assign it to the unique identifier <id>.

BITMAP <id> LOAD <filename$>

Load a Bitmap resource and assign it to the unique identifier <id>. <filename$> is the valid filename of a file in a supported Bitmap file format.

BITMAP <id> LOAD SHEET <filename$> [STRIP <extension$>]

Load a Sprite Sheet and assign it to the unique identifier <id>. <filename$> is the valid filename of a file in a supported Sprite Sheet file format. Sprite Sheets must be in XML format and define the full Bitmap to load in, as well as the names and coordinates for each subtexture in the Bitmap. The optional STRIP modifier will remove <extension$> from the end of each name if it is present.

BITMAP <id> SAVE <filename$>

Save the Bitmap <id> to disk in a supported Bitmap file format.

BITMAP <id> TARGET

Target the Bitmap <id> (must be between 1 and 1024). All subsequent Bitmap commands are directed at the targeted Bitmap.

BITMAP <id> RESCALE <width>,<height>

Rescale the Bitmap assigned to <id> to the width <width> and <height> specified. Cannot be used on Sprite Sheets.

BITMAP <id> FREE

Free a Bitmap from memory.

Texture Commands and Arguments

fill <color>

Create a new 256x256 Bitmap and fill it with <color>.

blobs <seed>,<amount>,<rgbFlag>

Create a new 256x256 Bitmap using Blobs. <rgb> is 1 for color, or 0 for greyscale.

cellmachine <seed>,<rule>

Create a new 256x256 Bitmap using a Cell Machine.

checkerboard <dx>,<dy>,<color1>,<color2>

Create a new 256x256 Bitmap using a Checkerboard pattern. <dx>,<dy> is the size (in pixels) of each block in the checkerboard. <color1> and <color2> are the colors to swap between.

mandelbrot <p1>,<p2>,<p3>,<p4>

Create a new 256x256 Bitmap using a Mandelbrot.

particle <size>

Create a new 256x256 Bitmap using a Particle. <size> is the size (in pixels) of the particle.

perlinnoise <dist>,<seed>,<amplitude>,<octaves>,<persistence>,<rgb>

Create a new 256x256 Bitmap using Perlin Noise. <rgb> is 1 for color, or 0 for greyscale.

sine <dx>,<dy>,<amplitude>

Create a new 256x256 Bitmap using Sine.

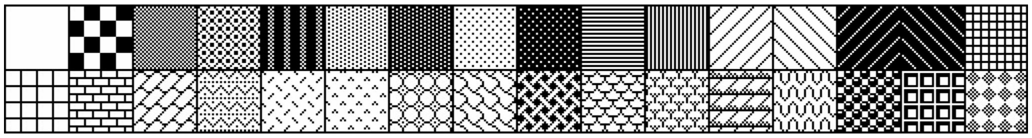
subplasma <dist>,<seed>,<amplitude>,<rgb>

Create a new 256x256 Bitmap using Sub Plasma.

Patterns

A pattern is a 32x32 Bitmap that is used when filling graphics primitives. Several Bitmap commands – like STRIP, RECT, CIRCLE, ELLIPSE, and FLOOD FILL – allow the shape to be filled with a solid color or a repeating pattern.

There are a total of 64 different Patterns. The first 32 Patterns are reserved and cannot be modified. They provide a variety of different bit patterns to use:



The second set of 32 Patterns (33 to 64) are user defined and can be customized to any 32x32 Bitmap.

Pattern Modifiers

PATTERN <id> USE BITMAP <bitmapId>

Set the Pattern <id> to <bitmapId>. <bitmapId> must be a 32x32 pixel Bitmap.

PATTERN <id> LOAD <filename$>

Load a Pattern (bitmap) resource and assign it to the unique identifier <id>. <filename$> is the valid filename of a file in a supported Bitmap file format. The Bitmap must be 32x32 pixels. (Use the BITMAP # RESCALE and PATTERN # USE BITMAP commands if needed to rescale the Bitmap to 32x32.)

PATTERN <id> SAVE <filename$>

Save the Pattern <id> to disk in a supported Bitmap file format.

PATTERN <id> TARGET

Target the Pattern <id> (must be between 33 and 64; <patternId's> between 1 and 32 are reserved and read-only). All subsequent Bitmap commands are targeted at the Bitmap that comprises the Pattern.

PATTERN <id> FREE

Free a Pattern from memory.

Layers

Layer Modifiers

LAYER <id> USE TILEMAP <tilemapId>

Use the TileMap <tilemapId> to render the Layer.

LAYER <id> USE BITMAP <bitmapId>

Use the Bitmap <bitmapId> to render the Layer.

LAYER <id> TARGET

Target the Layer <id> (must be between 1 and 8). If the Layer is TileMap-based, all subsequent TileMap commands are directed at the TileMap. If the Layer is Bitmap-based, all subsequent Bitmap commands are directed at the Bitmap.

LAYER <id> SHOW

Show (make visible) the Layer.

LAYER <id> HIDE

Hide (make invisible) the Layer.

LAYER <id> CENTER

Center the Layer in the screen. The aspect ratio is preserved, and the Layer is not scaled or stretched.

LAYER <id> STRETCH

Stretch the Layer to the screen. The aspect ratio is ignored.

LAYER <id> FILL

Fill the screen with the Layer while maintaining the aspect ratio. This may cause the Layer to be larger than the visible area of the screen.

LAYER <id> FIT

Fit the Layer to the screen while maintain the aspect ratio. This may cause the Layer to not fill the entire visible area of the screen.

LAYER <id> TILE

Tile the Layer. The aspect ratio is preserved, and the Layer is not scaled or stretched. It simply repeats to fill the screen.

LAYER <id> XYSIZE <xsize>,<ysize>

Set the Layer’s XYSIZE to <xsize>,<ysize>. Layers operate in 1280x800 mode. XYSIZE is useful to scale a Layer to “fit” or “fill” the screen.

LAYER <id> SCALE <sx>,<sy>

Scale the Layer horizontally/vertically by <sx>,<sy>. The scale is multiplied by XSIZE and YSIZE.

LAYER <id> SCALE TO <sx>,<sy>,<ms>

Scale the Layer horizontally/vertically by <sx>,<sy> over time. The scale is multiplied by XSIZE and YSIZE. <ms> is the number of milliseconds to scale the Layer. Multiple SCALE commands can be queued, so when one finishes the next one begins.

LAYER <id> ROTATE <degrees>

Rotate the Layer by <degrees> degrees.

LAYER <id> ROTATE TO <degrees>,<ms>

Rotate the Layer by <degrees> degrees over time. <ms> is the number of milliseconds to rotate the Layer. Multiple ROTATE commands can be queued, so when one finishes the next one begins.

LAYER <id> TINT <color>

Tint the Layer to <color>.

LAYER <id> TINT TO <color>,<ms>

Tint the Layer to <color> over time. <ms> is the number of milliseconds to tint the Layer. Multiple TINT commands can be queued, so when one finishes the next one begins.

LAYER <id> FLIP

Flip the Layer vertically.

LAYER <id> MIRROR

Mirror the Layer horizontally.

LAYER <id> FREE

Free a Layer from memory.

Sprites

Sprite Modifiers

SPRITE <id> USE TILEMAP <tilemapId>

Use the TileMap <tilemapId> to render the Sprite.

SPRITE <id> USE BITMAP <bitmapId>

Use the Bitmap <bitmapId> to render the Sprite.

SPRITE <id> SELECT <name$>

Use the Sprite Sheet's subtexture <name$> to render the Sprite.

SPRITE <id> SELECT ALL

Use the entire Bitmap to render the Sprite.

SPRITE <id> TARGET

Target the Sprite <id> (must be between 1 and 256). If the Sprite is TileMap-based, all subsequent TileMap commands are directed at the TileMap. If the Sprite is Bitmap-based, all subsequent Bitmap commands are directed at the Bitmap.

SPRITE <id> SHOW

Show (make visible) the Sprite.

SPRITE <id> HIDE

Hide (make invisible) the Sprite.

SPRITE <id> LAYER <layer>

Assign the Sprite to Layer <layer>. The Sprite will appear above the Layer and below the next Layer in the stack.

SPRITE <id> PRIORITY <priority>

Assign the Sprite's priority to <priority>. <priority> must be between 0 and 999999. The Sprite's priority determines the order the Sprites within a single Layer are drawn. Sprites with a higher priority appear on top of Sprites with a lower priority.

SPRITE <id> ANCHOR <anchorX>,<anchorY>

Assign the Sprite's anchor point to <anchorX>,<anchorY>. The anchor points must be between 0 and 1. The anchor point determines the "center" of the Sprite. By default, a Sprite's anchor point is the center (0.5, 0.5) of the TileMap or Bitmap the Sprite is using. When moving a Sprite to a coordinate this will center the Sprite on that coordinate. It also allows Sprites to be properly rotated around their center. Anchor points can be set to the upper left of the Sprite (0, 0) or the lower right (1, 1) or any point in-between.

SPRITE <id> XYSIZE <xsize>,<ysize>

Set the Sprite's XYSIZE to <xsize>,<ysize>. The XYSIZE is applied whenever a Sprite is scaled. Sprites operate in 1280x800 mode, regardless of what mode the Layer is in. XYSIZE is useful to fit a Sprite to a certain resolution (e.g. XYSIZE 4,4 to fit MODE 4's 320x200 Bitmap) and be scaled within that resolution.

SPRITE <id> XYSTEP <xstep>,<ystep>

Set the Sprite's XYSTEP to <xstep>,<ystep>. The XYSTEP is applied whenever a Sprite is moved. Sprites operate in 1280x800 mode, regardless of what mode the Layer is in. XYSTEP is useful to fit a Sprite to a certain resolution (e.g. XYSTEP 4,4 to fit MODE 4's 320x200 Bitmap) and be moved within that resolution.

SPRITE <id> MOVE <x>,<y>

Move a Sprite to <x>,<y>. The coordinates are multiplied by XSTEP and YSTEP.

SPRITE <id> MOVE RELATIVE <x>,<y>

Move a Sprite relative to its current position by <x>,<y>. The coordinates are multiplied by XSTEP and YSTEP.

SPRITE <id> MOVE TO <x>,<y>,<ms>

Move a Sprite to <x>,<y> over time. The coordinates are multiplied by XSTEP and YSTEP. <ms> is the number of milliseconds to move the Sprite. Multiple MOVE TO commands can be queued, so when one finishes the next one begins.

SPRITE <id> SCALE <sx>,<sy>

Scale the Sprite horizontally/vertically by <sx>,<sy>. The scale is multiplied by XSIZE and YSIZE.

SPRITE <id> SCALE TO <sx>,<sy>,<ms>

Scale the Sprite horizontally/vertically by <sx>,<sy> over time. The scale is multiplied by XSIZE and YSIZE. <ms> is the number of milliseconds to scale the Sprite. Multiple SCALE commands can be queued, so when one finishes the next one begins.

SPRITE <id> ROTATE <degrees>

Rotate the Sprite by <degrees> degrees.

SPRITE <id> ROTATE TO <degrees>,<ms>

Rotate the Sprite by <degrees> degrees over time. <ms> is the number of milliseconds to rotate the Sprite. Multiple ROTATE commands can be queued, so when one finishes the next one begins.

SPRITE <id> TINT <color>

Tint the Sprite to <color>.

SPRITE <id> TINT TO <color>,<ms>

Tint the Sprite to <color> over time. <ms> is the number of milliseconds to tint the Sprite. Multiple TINT commands can be queued, so when one finishes the next one begins.

SPRITE <id> FLIP

Flip the Sprite vertically.

SPRITE <id> MIRROR

Mirror the Sprite horizontally.

SPRITE <id> COLLISION CIRCLE <radius>

Sets the Sprite's collision circle radius to <radius>. The collision circle is the area of the Sprite that will trigger a collision when it "hits" another Sprite's collision circle. The default collision circle radius is either the width of the sprite divided in half or the height of the sprite divided in half, whichever is greater.

SPRITE <id> SHOW COLLISION CIRCLE

Show (make visible) the Sprite's collision circle. This command can be useful for debugging.

SPRITE <id> HIDE COLLISION CIRCLE

Hide (make invisible) the Sprite's collision circle. This command can be useful for debugging.

SPRITE <id> FREE

Free a Sprite from memory.

Sound

Sound Modifiers

SOUND <id> LOAD <filename$>

Load a Sound resource and assigns it to the unique identifier <id>. <id> must be between 1 and 256. <filename$> is the valid filename of a file in a supported Sound file format.

SOUND <id> PLAY

Play a Sound starting at the beginning.

SOUND <id> PAUSE

Pause a Sound that is playing.

SOUND <id> RESUME

Resume a Sound that was paused.

SOUND <id> STOP

Stop a Sound that is playing.

SOUND <id> VOLUME <volume>

Set the volume of the Sound.

SOUND <id> FREE

Free a Sound from memory.

Voice

Voice Modifiers

VOICE SAY [ASYNC] <speech$>

Say (speak) the text in <speech$> using the current selected voice. Use the optional ASYNC modifier to continue program execution while the computer is talking.

VOICE SELECT <voice$>

Select the voice <voice$> to use when speaking. Use the INSTALLEDVOICES$ reserved variable to get a list of available voices.

The Standard Library

The Standard Library is a collection of constants, reserved variables, and built-in functions that provide math, string, time, keyboard, mouse, and other useful routines.

Constants and reserved variables can be read but not assigned.

Reserved Constants

VERSION$

Returns the version of Liquid BASIC in use.

FALSE

Return 0.

TRUE

Return 1.

EMPTY$

Returns an empty (“”) String.

PI

Return 3.1415926535897931.

FLICK

Return 1.417233560090703^{-9}.

Reserved Variables

TIMER

Return the number of seconds since midnight.

TIME

Return the number of milliseconds since Liquid BASIC started.

ELAPSEDTIME

Return the number of milliseconds since ELAPSEDTIME was last read.

TIME$

Return the current time in HH:MM:SS format.

DATE$

Return the current date in MM/DD/YYYY format.

GUID$

Return a unique GUID.

ERR

Return the last error code that is recorded.

ERL

Return the line number of the last error that is recorded.

INKEY$

Return the next key in the keyboard buffer, or “” if the keyboard buffer is empty.

FREEFILE

Return the next valid unused file handle, or 0 if there are no free file handles available. This is useful to avoid having FUNCTIONs use file handles that are already in use.

CURDIR$

Return the current directory.

ROOTDIR$

Return the program’s root directory.

CURSORX

Return the current cursor X position.

CURSORY

Return the current cursor Y position.

MOUSEX

Return the current mouse X position.

MOUSEY

Return the current mouse Y position.

MOUSEWHEEL

Return the next mouse wheel event in the mouse event buffer, or 0 if the mouse event buffer is empty.

MOUSEBUTTON

Return 1 if the left mouse button is down, 2 if the right mouse button is down, 3 if both buttons are down, or 0 if no mouse button is down.

SCREENWIDTH

Return the screen width in pixels (1280).

SCREENHEIGHT

Return the screen height in pixels (800).

FPS

Return the number of Frames Per Second being rendered.

TEXTCOLUMNS

Return the number of columns in the text screen.

TEXTROWS

Return the number of rows in the text screen.

FREETILESET

Return the next valid unused TileSet ID, or 0 if there are no free TileSets available.

FREETILEMAP

Return the next valid unused TileMap ID, or 0 if there are no free TileMaps available.

FREEFONT

Return the next valid unused Font ID, or 0 if there are no free Fonts available.

FREEBITMAP

Return the next valid unused Bitmap ID, or 0 if there are no free Bitmaps available.

FREEPATTERN

Return the next valid unused Pattern ID, or 0 if there are no free Patterns available.

FREELAYER

Return the next valid unused Layer ID, or 0 if there are no free Layers available.

FREESPRITE

Return the next valid unused Sprite ID, or 0 if there are no free Sprites available.

FREESOUND

Return the next valid unused Sound ID, or 0 if there are no free Sounds available.

RNDCOLOR

Return a random color.

PLASMACOLOR

Return the constantly evolving plasma color.

INSTALLEDVOICES$

Return a comma separated list of the available voices for speech.

COPILOT

Return 1 if OpenAI is present. Otherwise return 0.

CHATERROR$

Return the last error generated by OpenAI. Otherwise return an empty string (“”).

Error Handling

ERR$(<code>)

Convert an error code to a String.

Conversion Functions

VAL(<d$>)

Convert a String to a Real and return.

STR$(<d>)

Convert a Real to a String and return.

BIN$(<d>)

Convert a Real to an integer and return as a String in base 2 (binary) format.

HEX$(<d>)

Convert a Real to an integer and return as a String in base 16 (hexadecimal) format.

OCT$(<d>)

Convert a Real to an integer and return as a String in base 8 (octal) format.

Numeric Functions

ABS(<value>)

Return the absolute value of <value>.

ATN(<value>)

Return the arctangent of <value>.

CEIL(<value>)

Round <value> up and return.

CLAMP(<value>,<min>,<max>)

Return <value> clamped to <min> and <max>.

COS(<value>)

Return the cosine of <value>.

DEG(<value>)

Return the radians <value> converted to degrees.

EXP(<value>)

Return the exponent of <value>.

FLOOR(<value>)

Round <value> down and return.

INT(<value>)

Return the integer portion of <value>.

LOG(<value>)

Return the logarithm of <value>.

MIN(<x1>,<x2>[,<x3>…])

Return the minimum value out of the arguments passed.

MAX(<x1>,<x2>[,<x3>…])

Return the maximum value out of the arguments passed.

RAD(<value>)

Return the degrees <value> converted to radians.

RANGE(<start>,<end>)

Return a random integer number between <start> and <end>.

RND([<type>])

Return a random number between 0 and 1 if <type> is not zero or omitted, otherwise return the last random number generated.

ROUND(<value>[,<digits>])

Round <value> to the specified number of <digits> and return. If <digits> is omitted, round to the nearest integer.

SGN(<value>)

Return the sign of <value> (-1 if negative, 0 if zero, 1 if positive).

SIN(<value>)

Return the sine of <value>.

SQR(<value>)

Return the square root of <value>.

TAN(<value>)

Return the tangent of <value>.

String Functions

ASC(<string>[,<index>])

Return the ASCII code of the character at the first position in string, or the character at the <index> position if specified. Character positions in Strings start at 1 in Liquid BASIC.

CHR$(<ch>[,<ch>…])

Return a string with the ASCII character <ch>. Multiple characters can be included.

CONCAT$([<x1>[,<x2>]…])

Take zero, one, or more arguments (Real or String) and concatenate them into a single String.

EXPORT$([<x1>[,<x2>]…])

Take zero, one, or more arguments and combine them into a comma separated list. Real values are converted to a String value, and String values are wrapped in quotes (to preserve commas). Complements the PARSE$ and PARSECOUNT functions.

HASH(<text$>)

Return a 32-bit unsigned integer hashed from string <text$>.

INSTR([<startindex>,]<text$>,<value$>)

Return the index where the string <value$> is found in the string <text$>, or 0 if <value$> was not found. The optional <startIndex> is the index to start searching.

LCASE$(<text$>)

Return the string <text$> converted to lowercase letters.

LEFT$(<text$>,<length>)

Return the left side of the string <text$>, up to <length> characters.

LEN(<text$>)

Return the length of the string <text$>.

LTRIM$(<text$>[,<char$>])

Return the string <text$> with all leading <char$> removed (if <char$> is omitted then all leading spaces are removed).

MID$(<text$>,<position>[,<length>])

Return a substring from <text$> starting at <position>. Up to <length> characters are returned if <length> is specified. Otherwise, the remainder of the string <text$> is returned.

PARSE$(<text$>,<index>)

Return a single item from the comma separated list in <text$> at index <index>. Complements the EXPORT$ function.

PARSECOUNT(<text$>)

Return the number of items in the comma separated list in <text$>. Complements the EXPORT$ function.

REGEXPR([<startindex>,]<text$>,<pattern$>)

Return the index where the regular expression <pattern$> is found in the string <text$>, or 0 if <pattern$> was not found. The optional <startindex> is the index to start searching.

REGREPL$(<text$>,<pattern$>,<replacement$>)

Return the string <text$> but replace anywhere the regular expression <pattern$> is found with <replacement$>.

RIGHT$(<text$>,<length>)

Return the right side of the string <text$>, up to <length> characters.

RTRIM$(<text$>[,<char$>])

Return the string <text$> with all trailing <char$> removed (if <char$> is omitted then all trailing spaces are removed).

SPACE$(<count>)

Return a string of spaces repeated <count> times.

STRDELETE$(<text$>,<index>,<count>)

Return the string <text$> with <count> characters deleted starting at <index>.

STRINSERT$(<text$>,<insert$>,<index>)

Return the string <text$> with the string <insert$> inserted at <index>.

STRREPEAT$(<text$>,<count)>

Return a string with <text$> repeated <count> times.

STRREPLACE$(<text$>,<value$>,<replacement$>)

Return the string <text$> but replace anywhere the string <value$> is found with <replacement$>.

STRREVERSE$(<text$>)

Return the string <text$> reversed.

TRIM$(<text$>[,<char$>])

Return the string <text$> with all leading and trailing <char$> removed (if <char$> is omitted then all leading and trailing spaces are removed).

UCASE$(<text$>)

Return the string <text$> converted to uppercase letters.

USING$(<format$>,<value>)

Return <value> as a String formatted to the format list <format$>. <value> can be a Real or a String.

FORMAT LIST

For Real values, the pound sign '#' reserves room for a single character in the output field. If the formatted string contains more characters than there are pound signs in <format$>, then the entire output field is filled with asterisks '*'.

- If <format$> starts with a plus sign '+', then a plus sign will be printed at the beginning of the output field when the value is positive, and a minus sign will be printed at the beginning of the output field when the value is negative.
- If <format$> starts with a minus sign '-', then a space will be printed at the beginning of the output field when the value is positive, and a minus sign will be printed at the beginning of the output field when the value is negative.
- If <format$> does not start with either a plus sign or a minus sign, then a minus sign is printed before the first formatted character for negative values, taking up one character in the output field. Remember that if the number of characters (including the minus sign) exceeds the size of the output field the entire output field is replaced with asterisks.
- If <format$> starts with a dollar sign '$', then the output field will begin with a dollar sign and positive and negative values will be printed using the rule above.
- A period '.' in <format$> is used to represent where the decimal point, if any, should appear. There cannot be more than one decimal point. If no decimal point is specified the value is rounded to the nearest integer and printed without a decimal point.
- A comma in <format$> is used to represent where commas, if any, should appear.

For String values, the pound sign '#' reserves room for a single character in the output field. If the formatted string contains more characters than there are pound signs in <format$>, then the formatted string is left justified and truncated to fit the output field.

- If <format$> starts with the less than sign '<', then the formatted output is left justified.
- If <format$> starts with the equal sign '=', then the formatted output is centered within the output field.
- If <format$> starts with the greater than sign '>', then the formatted output is right justified.

File Functions

DIR$([<filter>[,<mode>]])

Return a comma separated list depending on the <mode>:

1	Return a list of files in the current directory
2	Return a list of directories in the current directory
3	Return a list of both files and directories in the current directory (default)

The results are filtered to <filter$>. Wildcard characters like the asterisk '*' and question mark '?' can be used. The filter defaults to "*.*" if it is omitted.

EXISTS(<path$>)

Return 1 if the file or directory <path$> exists. Otherwise return 0.

EOF(<handle>)

Return 1 if at the end of the file assigned to <handle>. Otherwise return 0.

LOC(<handle>)

Return the position (in bytes) of the file assigned to <handle>.

LOF(<handle>)

Return the length (in bytes) of the file assigned to <handle>.

FILEOPEN$(<directory$>,<extension$>,<filter$>)

Open the host OS's File Open dialog and return the selected file (or "" if no file was selected). <directory$> is the initial directory to start in (use "default" to restore to the last used directory). <extension$> is the default file extension. <filter$> is the filename filter string, which determines the choices that can be selected as a file type.

FILESAVEAS$(<directory$>,<extension$>,<filter$>)

Open the host OS's File Save As dialog and return the selected file (or "" if no file was selected). <directory$> is the initial directory to start in (use "default" to restore to the last used directory). <extension$> is the default file extension. <filter$> is the filename filter string, which determines the choices that can be selected as a file type.

Database Functions

DBERROR$(<handle>)

Return the error message from the last DB SCALAR, DB QUERY, DB READ, or DB WRITE command run on the database assigned to <handle> if there was an error. Otherwise return an empty string ("").

DBNAMES$(<handle>)

Return a comma separated list of the column names in the last query run on the database assigned to <handle>.

EOQ(<handle>)

Return 1 if at the end of the query in the database assigned to <handle>. Otherwise return 0 if there are more query results left to read with the DB READ command.

Keyboard Functions

KEYDOWN(<key>)

Return 1 if the <key> is currently down. Otherwise return 0. Several constants are included to simplify the key codes:

	CONSTANT	**KEY**
1		(nothing)
2	VK_INSERT	Insert
3	VK_DELETE	Delete
4	VK_HOME	Home
5	VK_END	End
6	VK_PAGEUP	Page Up
7	VK_PAGEDOWN	Page Down
8	VK_BACKSPACE	Backspace
9	VK_TAB	Tab
10	VK_LINEFEED	Line Feed
11	VK_SHIFTTAB	SHIFT + Tab

12		(nothing)
13	VK_ENTER	Enter
14	VK_LEFT	Left
15	VK_RIGHT	Right
16	VK_UP	Up
17	VK_DOWN	Down
18	VK_F1	F1
19	VK_F2	F2
20	VK_F3	F3
21	VK_F4	F4
22	VK_F5	F5
23	VK_F6	F6
24	VK_F7	F7
25	VK_F8	F8
26	VK_F9	F9
27	VK_ESC	Esc
28	VK_F10	F10
29	VK_F11	F11
30	VK_F12	F12
31		(nothing)
32	VK_SPACE	Space

Values 33 to 126 for <key> map to their respective ASCII character.

KEYUP(<key>)

Return 1 if the key <key> is currently up. Otherwise return 0.

KEYPRESSED(<key>)

Return 1 if the key <key> was pressed. Otherwise return 0.

Color Functions

HSV(<h>,<s>,<v>[,<a>])

Return a color using the hue <h>, saturation <v>, brightness <v>, and optional <a> components given. If <a> is omitted the alpha value is assumed to be 255 (for a solid, opaque color).

RGB(<r>,<g>,<b>[,<a>])

Return a color using the red <r>, green <g>, blue <b>, and optional alpha <a> components given. If <a> is omitted the alpha value is assumed to be 255 (for a solid, opaque color).

GETRED(<color>)

Return the red byte of the color <color>.

GETGREEN(<color>)

Return the green byte of the color <color>.

GETBLUE(<color>)

Return the blue byte of the color <color>.

GETALPHA(<color>)

Return the alpha byte of the color <color>.

CLUT(<name$>,<index>)

Lookup a color in one of the available CLUTs (Color LookUp Tables): C64 (32 colors), Amiga (32 colors), VGA (256 colors), or ViceCity (16 colors). <name$> is the CLUT to lookup, and <index> is the index of the color to get.

COLORPICKER(<color>)

Open the host OS's Color Picker dialog and return the selected color (or <color> if no color was selected).

TileMap Functions

TILEMAPCOLUMNS([<id>])

Return the number of columns in TileMap <id>, or the number of columns in the active TileMap if <id> is omitted.

TILEMAPROWS([<id>])

Return the number of rows in TileMap <id>, or the number of rows in the active TileMap if <id> is omitted.

CHAR(<ch>)

Return the tile mapped to the character <ch>.

CHAR(<text$>)

Return the tile mapped to the character in <text$>. <text$> must be a string exactly one character in length.

CHAR$(<text$>)

Return the tiles mapped to the characters in <text$>.

PEEKTILE(<index>)

Return the tile at <index>. Complements the POKE TILE command.

PEEKINK(<index>)

Return the ink color at <index>. Complements the POKE INK command.

PEEKHIGHLIGHT(<index>)

Return the highlight color at <index>. Complements the POKE HIGHLIGHT command.

GETTILE(<x>,<y>)

Return the tile at <x>,<y>. Complements the PUT TILE command.

SHIFTX([<id>])

Return the current X shift in pixels in TileMap <id>, or the current X shift in pixels in the active TileMap if <id> is omitted.

SHIFTY([<id>])

Return the current Y shift in pixels in TileMap <id>, or the current Y shift in pixels in the active TileMap if <id> is omitted.

Bitmap Functions

BITMAPWIDTH([<id>])

Return the width in pixels of Bitmap <id>, or the width in pixels of active Bitmap if <id> is omitted.

BITMAPHEIGHT([<id>])

Return the height in pixels of Bitmap <id>, or the height in pixels of active Bitmap if <id> is omitted.

PEEK(<index>)

Peek (read directly from memory) the pixel at <index> and return. Complements the POKE command.

POINT(<x>,<y>)

Return the pixel color at <x>,<y>. Complements the PLOT command.

SAMPLE(<x>,<y>)

Return the pixel color at <x>,<y>, where both <x> and <y> are normalized to 0 <= x <= 1. For example, the coordinates 0.5, 0.5 would refer to the pixel at the center of the Bitmap.

COPY$(<x1>,<y1>,<x2>,<y2>)

Copy the rectangular Bitmap data from <x1>,<y1> to <x2>,<y2> and return it in an encoded string. Complements the PASTE command.

SHEETNAMES$(<id>)

Return a comma separated list of the subtexture names available in Bitmap <id>.

Sprite Functions

SPRITEX(<id>)

Return the x-position of Sprite <id>.

SPRITEY(<id>)

Return the y-position of Sprite <id>.

INSIDE(<id1>,<id2>)

Return 1 if Sprite <id1> is inside Sprite <id2>. Otherwise return 0.

INTERSECT(<id1>,<id2>)

Return 1 if Sprite <id1> is intersecting Sprite <id2>. Otherwise return 0.

TOUCH(<id1>,<id2>)

Return 1 if Sprite <id1> is touching Sprite <id2>. Otherwise return 0.

HOVER(<id>)

Return 1 if the mouse is hovering over Sprite <id>. Otherwise return 0.

CLICKED(<id>)

Return 1 if the mouse is clicked on Sprite <id>. Otherwise return 0.

Error Codes

Liquid BASIC uses error codes to record an error. Error codes are stored in the ERR reserved variable. The ERR$ function can be used to convert an error code to text.

1	Unknown
2	Assertion Failed
3	Input Statement Not Allowed
4	Print Statement Not Allowed
5	End Of Line Expected
6	Unexpected End Of Program
7	Closing Quote Expected
8	Invalid Hex Data
9	Invalid Delimiter
10	Invalid Operator
11	Syntax
12	Valid Line Number Expected
13	Valid Line Range Expected
14	Valid Line Increment Expected
15	Duplicate Label
16	Cannot Find Label
17	Illegal Direct
18	Illegal Quantity
19	Duplicate Identifier
20	Undeclared Variable
21	Variable Is Read Only
22	Variable Is Array
23	Variable Is Not Array
24	Array Not Dimensioned
25	Array Already Dimensioned
26	Array Is Not One Dimension
27	Bad Subscript
28	Type Mismatch
29	Division By Zero
30	String Too Long
31	Out Of Data
32	Bad GOTO
33	Bad RESTORE
34	Bad DATA
35	Bad TRAP

36	GOSUB Stack Overflow
37	GOSUB Stack Empty
38	CALL Stack Overflow
39	CALL Stack Empty
40	RETURN Without GOSUB
41	IF Without END IF
42	ELSE Without IF
43	END IF Without IF
44	EXIT IF Without IF
45	SELECT Without END SELECT
46	CASE Without SELECT
47	END SELECT Without SELECT
48	EXIT SELECT Without SELECT
49	WHILE Without WEND
50	WEND Without WHILE
51	EXIT WHILE Without WHILE
52	DO Without LOOP
53	LOOP Without DO
54	EXIT DO Without DO
55	FOR Without NEXT
56	NEXT Without FOR
57	EXIT FOR Without FOR
58	RESUME Without TRAP
59	Undeclared Function
60	Cannot Nest Functions
61	Invalid Number Of Arguments
62	EXIT FUNCTION Without FUNCTION
63	END FUNCTION Without FUNCTION
64	Valid Modifier Expected
65	Valid Filename Expected
66	File Not Found
67	File Open
68	File Not Open
69	Access Denied
70	Unable To Open File
71	Unable To Seek In File
72	Unable To Get From File
73	Unable To Put To File
74	Unable To Read From File
75	Unable To Write To File
76	Unable To Input From File
77	Unable To Print To File
78	Unable To Close File
79	Unable To Move File
80	Unable To Copy File
81	Unable To Delete File
82	Directory Not Found
83	Directory Already Exists
84	Unable To Change Directory

85	Unable To Make Directory
86	Unable To Remove Directory
87	Database Open
88	Database Not Open
89	Unable To Open Database
90	Unable to Close Database
91	No Palette
92	No TileSet
93	No TileMap
94	No Font
95	No Bitmap
96	No Pattern
97	No Layer
98	No Sprite
99	No Sound
100	No Voice
101	Color Not Found
102	Tile Too Small
103	Tile Too Big
104	TileSet Is Read-Only
105	TileMap Too Small
106	TileMap Too Big
107	TileMap Single Buffer
108	Bitmap Too Small
109	Bitmap Too Big
110	Bitmap Single Buffer
111	Pattern Is Read-Only
112	Pattern Wrong Size
113	Duplicate Subtexture
114	Subtexture Not Found
115	Modifier Not For Subtextures
116	No Copilot
117	Illegal Operation
118	Unable To Generate Image