Dragon Basic manual

Dragon Basic
GBA Development

Documentation

van Zoelen A.A.

Document details		**Dragon Basic**	
Version:	1.01	Author	: A.A. van Zoelen
Creation Date:	17 June 2003	Reviewer	: Jeff Massung
Print Date	{printdate * Mergeformat }		

This page has intentionally been left blank.

1. General

1.1 What is this?

This document is created as a manual for Dragon Basic. Dragon Basic is a compiler for GBA development.

1.2 Index

{ TOC \o "1-3" }

1.3 Introduction

This manual will provide background information about the Dragon Basic compiler, the language used and GBA development in general.

1.4 What is "Dragon Basic"

Dragon Basic is an implementation of the Basic programming language in such way that the user is able to program the GBA. The Gameboy Advance is a handheld game console made by Nintendo ®™. While the development on the GBA is a 'closed' environment and only possible with expensive licenses and equipment bought by Nintendo it is possible to create game ROMS for the so called 'homebrew' market. However Nintendo doesn't endorse this development.

Dragon Basic makes use of the Basic language syntax to give the programmer an easy development environment. Because Basic is a simple language to learn it will be open for everyone to compile their dream into a ROM image. This ROM image can be tested in an emulator or directly on the hardware via various ways. (That beyond this manual)

Dragon Basic is created by Jeff Massung

Tip 1 *Emulator are freely found on the Internet. Search for "GBA emulator" for example.*

Tip 2 *Check* ***{ HYPERLINK http://www.simforth.com }*** *for the latest version of the compiler.*

Tip 3 *Check* ***{ HYPERLINK http://www.simforth.com }*** *for the latest info and support forum*

2 Getting started

2.1 Installation

Dragon Basic comes in two forms. With, and without an installer. When using the installer just follow the instruction on screen. Because the compiler is just a single file there isn't really need for using an installer. Just create a folder to hold the DBC.exe and the constants definition file called GBA.dbc That's it, nothing more is needed.

2.2 Creating your code

To compile something you first need some code to compile. There are various ways to create your code as long as the output is just a plain ASCII text file. I can name many different editor but I just stick with NotePad.exe as an example.

Tip *The { HYPERLINK http://www.simforth.com } web site holds several clear examples and tutorials.*

2.3 Compiling

After you created your program, save it to disk. One of the ways to get you code compiled is to open a command box and type **dbc <source filename> <compiled filename>** and hit enter. That will put the compiler at work and a ROM images is created suitable to be used on the real hardware or in an emulator.

3 The commandset – Compiler commands

There are three types of commands in Basic so this is in Dragon Basic also the case. They are compiler commands, commands that do things and commands/keywords that are a part of the syntax. I will handle them as three separate groups. Then in the next section i will group then alphabetically. And in the last section they are ordered by functionality. The alphabetically sorted section (chapter 4) describes the commands in detail. The other chapters are for extra information.

Compiler commands are commands that tell the compiler to do something specific. While other commands will do this to, the difference here is that these command give instructions to the compiler to shape the output in a different form then the default form or to add extra information to the ROM image.

3.1 #ALIGN bytes

Aligns the ROM binary along a bytes boundary. This is good to do before importing data that must be aligned specially, or if you want to know where a section of code where end up.

The compiler will not automatically word align before importing data (with **#IMPORT**, **#BITMAP**, **#PALETTE** or **#SOUND**), but will after importing. It will automatically align before a **#POOL** directive is compiled (whether directly or indirectly called).

Return value: None.

bytes	The size of the boundary to align on

```
; Make sure the data is page aligned

#align 256
data 1,2,3,4
```

3.2 #BITMAP filename

Extracts and compiles into the ROM either the pixel data from a 24-bit image file, or the tile data from a 4-bit or 8-bit image file.

The image file must be a PNG or PCX file. It must be 4, 8 or 24 bit pixel depth. And for a sprite or tile, it must be a multiple of 8x8 pixels in size.

Return value: None.

filename	A string that evaluates to a path and filename

```
; Create a label where an image is loaded into ROM

my_image: #bitmap "background.pcx"
```

3.3 #CONSTANT symbol value

Creates a new constant symbol with a literal value. Value can be a literal value (ie 45), a string, a label or another constant, but not a variable or expression.

Return value: None.

symbol	Any valid variable name
Value	A literal, string, label or constant

```
; Create some constants

#constant msg "Hello, world!"
#constant four 4
```

3.4 #FONT string

All strings in Dragon BASIC use a lookup table, built into the compiler, that will act as an ASCII table.

#FONT allows you to overwrite the table the compiler uses with a different one. Note: to use the **SCORE** function, you *must* have the characters ' 0' -' 9' in your font table in sequential order.

Return value: None.

string	A string of ASCII characters

```
; Create a font table of just letters and numbers

#font "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
```

3.5 #IMPORT Filename

Imports a binary file (byte for byte) into your ROM. No data aligning is performed before this, so if necessary, you may want to use **#ALIGN** before **#IMPORT**.

Return value: None.

filename	A string that evaluates to a path and filename

```
; Import some compiled THUMB assembly that you can {GOTO}

my_thumb_routine: #import "code.bin"
```

3.6 #INCLUDE Filename

Includes the source code filename into your program, compiling it before continuing to compile the current file.

Return value: None.

filename	A string that evaluates to a path and filename

```
; Include the GBA.DBC constants file

#include "../gba.dbc"
```

3.7 #PALETTE Filename

#PALETTE will copy any palette data from a 4-bit or 8-bit image file. If the image has an optimized palette (does not use all the available color indices) it will pad them with white.

The image file must be a PNG or PCX file.

Return value: None.

filename	A string that evaluates to a path and filename

```
; Load a palette from a PNG file into ROM and then RAM

img_pal: #palette "sprite.png"
start:
  loadpal16 SPRITE_PALETTE,0,img_pal
```

3.8 #POOL

The ARM processor (in THUMB mode) cannot set registers with literal values > 255. It must load them from ROM. When doing this, there is a byte range limit of how far away the literal can be from the expression in code.

The **#POOL** directive is used as a placeholder, telling the compiler to place those values "here".

The compiler will automatically execute a **#POOL** directive after an infinite **WHITE/LOOP**, a "unburied" **GOTO** statement, an "unburied" **RETURN** statement, **END FUNCTION**, or **END**.

Return value: None.

None	

```
; Set a variable to a value

x = $1FF

; $1FF > 255, we need a #POOL directive

#pool
```

3.9 #SOUND filename

Extracts and converts an 8 or 16-bit, mono or stereo, PCM WAV file into signed, 8-bit mono PCM data that the GameBoy Advance can read and compiles it into ROM.

Return value: None.

filename	A string that evaluates to a path and filename

```
; Create a label pointing to a sound file
```

```
boom: #sound "boom.wav"

; Play it
start:
  playsound boom
```

3.10 #TITLE name

Sets the 12-character name of your game in the compiled ROM header. It can be more or less than 12 characters, but name will be padded with spaces if less and truncated at 12 if longer.

Return Value: none

name	An ASCII character string

```
; Set the title of my game:

#title "breakout"
```

4 The commandset – Sorted alphabetically

This section shows the commands in alphabetical order. The compiler commands are left out of it. See section 3 for more detail on them.

4.1 ABS (n)

Computes the absolute value of n. n can be an integer or fixed-point value.

Return value: The absolute value of n

n	n integer or fixed-point value

```
; ABS Example Code

x = Abs(-10) ; x = 10
x = Abs(2) ; x = 2
x = Abs(-2.5) ; x = 2.5
```

4.2 ALSO

A logical AND (ie. $F0 ALSO $0F = TRUE /* boolean */).

Return value: Boolean

none	Boolean

```
; ALSO Example Code

$F0 ALSO $0F (the result will be a TRUE)
```

4.3 AND

A bitwise AND

Return value: Integer

none	

```
; AND Example Code

$F0 AND $0F = $00

     1001  0111  0000
AND  0011  1111  1111
     ----  ----  ----
     0001  0111  0000
```

4.4 ANIMSPRITE sprite,first,last,blocks

Sets the current animation frame for sprite by either initializing it at first frame, or incrementing it by blocks to the next frame. It will reset back to the first frame if it exceeds the last frame.

Return value: None

sprite	Number of the sprite to animate (0-127)
First	The character block of the first frame
Last	The character block of the last frame
blocks	The number of character blocks to increment each frame

```
#constant my_sprite 0
#constant first 0
#constant last 24
#constant blocks 8
#constant stand 32

; Make a sprite animate for 50 frames

for i = 1 to 50
  animsprite my_sprite,first,last,blocks
next

; Reset the sprite to be still

animsprite my_sprite,stand,stand,0
```

4.5 BLIT screen,address,x,y,width,height

Pastes the bitmap image at address to screen at coordinates (x,y) of width and height. It will not copy over pixel values that are black (%000000000000000). This allows you to make an image have a transparent mask.

Return value: None.

screen	Video RAM address
address	Source memory address
x	The X coordinate where to paste
y	The Y coordinate where to paste
width	Width of the image
height	Height of the image

```
; Import an image

img: ; 45x67, 24-bit image
  #bitmap "my_img.pcx"

; BLIT it to the screen
start:
  graphics 3,0
  blit SCREEN,img,10,10,45,67
```

4.6 BLOCKS (width,height,depth)

Use BLOCKS to calculate how many blocks of RAM an image will take up.

Return value: Number of blocks used.

width	width of the image in pixels
height	height of the image in pixels
depth	pixels depth (4 or 8)

```
; Load a tile map that is 64x64 and 8-bit into RAM:

loadtiles charblock(0),my_map,blocks(64,64,8)
```

4.7 BUMPSPRITES (sprite1,sprite2)

Checks to see if two sprites are occupying the same space on or off screen. This is a bounding-box check, and *not* pixel perfect.

Return value: 0 if they do not overlap, non-zero if they do.

sprite1	a sprite (0-127)
sprite2	a sprite (0-127)

```
; Check to see if MARIO collided with YOSHI:

if bumpsprites(MARIO,YOSHI)
  goto die
end if
```

4.8 CHARBLOCK (n)

Calculates the address of character base block n.

Return value: The address of character base block n.

n	A character base block from 0-3

```
; Load some tiles into RAM

loadtiles charblock(2),my_tiles,45
```

4.9 CIRCLE buffer,x,y,radius,color

Draws a Bresenham circle of color with an origin at (x,y) of radius in pixels on the screen buffer in graphics modes 3 or 5.

Return value: None

buffer	Video RAM address
x	X coordinate of the origin
y	Y coordinate of the origin
radius	Radius (in pixels) of the circle
color	A 15-bit color value in BBBBBGGGGGRRRRR format

```
; Draw a red circle on the screen

graphics 3,0
circle SCREEN,120,80,40,RED
```

4.10 CLEARTILES tile,width,height

Erases all the tile data of a background starting at tile in an area bounded by width and height. All tiles in this area will be set to tile 0.

Return value: None.

tile	A tile address
width	Number of tiles in X direction
height	Number of tiles in Y direction

```
; Clear all the tiles in a screen block

cleartiles tile(4,0,0),32,32
```

4.11 CLOCKTIMER

Reads the number of times the current timer has fired.

Return value: The number of fires since the last reset.

None.	

```
; Time how long a loop takes to execute

maketimer 1000 ; fire every millisecond

; Loop 1000 times
for i = 1 to 1000
  ; Do nothing
next

; How long did that take?
i = clocktimer
```

4.12 CLS buffer,color

Erases the screen buffer with color in graphics modes 3 or 5.

Return value: None

buffer	Video RAM address
color	A 15-bit color value in BBBBBGGGGGRRRRR format

```
; Make the GBA screen blue

graphics 3,0
cls SCREEN,BLUE
```

4.13 COLORSPRITE sprite,palette

Sets the sprite to 16-color mode and selects the palette index to use. If your sprite is 4-bit, you *must* call this function after **MAKESPRITE** for your sprite to draw properly.

Return value: None.

sprite	A sprite (0-127)
palette	A 16-color palette index (0-15)

```
; Make a 16x32 4-bit sprite and use palette # 1

makesprite 0,512
sizesprite 0,TALL,SIZE_32
colorsprite 0,1
```

4.14 COLORTILE address,palette

Changes the palette used by the tile at address. This function has no effect if the background is in 256 color mode.

Return value: None.

address	An address gotten with TILE
palette	A palette index (0-15)

```
; Change the palette used by the tile at 10,12 of
screenblock 4.

colortile tile(4,10,12),5
```

4.15 COPY dest,source,words

Copies words (32-bits) of data from source address to dest address.

Return value: None.

dest	Destination address
Source	Source address
Words	Number of 32-bit values to copy

```
; Create two arrays of data

global array_a(10)
global array_b(10)

; Copy all the values from b into a

copy array_a&,array_b&,10
```

4.16 COS (degrees)

Computes the cosine of an angle in degrees between 0 and 359.

Return value: A fixed-point value that is the cosine of degrees.

degrees	An integer angle between 0 and 359

```
; Compute the cosine of 45

x = cos(45) ; x = 0.707
```

4.17 DISABLEMOSAIC background

Toggles off the mosaic bit so that a background layer (0-3) will not be affected by calls to **MOSAIC**.

Return value: none.

background	the background layer (0-3)

```
; Turn off the mosaic bit for background 2

disablemosaic 2
```

4.18 DISABLETILES background

Disables the bit in REG_DISPCNT for a background layer so that it is no longer drawn.

Return value: none.

background	a background layer (0-3)

```
; Disable background 1

disabletiles 1
```

4.19 ENABLEMOSAIC background

Toggles on the mosaic enable bit for a background' s control register.

Return value: none.

background	the background layer to enable mosaic effect (0-3)

```
; Turn on the mosaic enable bit for background 2

enablemosaic 2
```

4.20 ENABLETILES background,screen,char,flags

Sets the background layer bit in REG_DISPCNT so that a layer of tiles is drawn.

It also sets the screen block and character block for a group of tiles to use.

It will also set any flags (i.e. BG_COLOR_16) that are passed to it.

background	the background to enable (0-3)
screen	the screen block to use (0-31)
char	the character block to use (0-3)
flags	any extra flags to set (or'ed together)

```
; Setup background 0 to be drawn, use screenblock 8 and
charblock 0

enabletiles 0,8,0,BG_COLOR_256 or TEXT_SIZE_256x256
```

4.21 ERASE address,words

Zeroes words (32-bit) of data at address.

Return value: None.

address	Destination address
words	Number of 32-bit values to zero

```
; Create an array

global array(100)

; Zero out all data in it

erase array&,100
```

4.22 FADD (n1,n2)

Adds the two 16:16 fixed point values n1 and n2.

Return value: An 16:16 fixed point value (n1+n2).

n1	A 16:16 fixed point value
n2	A 16:16 fixed point value

```
; Add two fixed-point values

x = fadd(1.2,2.3) ; x = 3.5
```

4.23 FDIV (n1,n2)

Divides the two 16:16 fixed point values n1 and n2.

Return value: An 16:16 fixed point value (n1/n2).

n1	A 16:16 fixed point value
n2	A 16:16 fixed point value

```
; Divide two fixed-point values

x = fdiv(10.0,2.5) ; x = 4.0
```

4.24 FIX (n)

Converts n - a 32-bit integer to a 16:16 fixed point number.

Return value: A 16:16 fixed point value.

n	A 32-bit integer value

```
; Convert an integer to a fixed-point value

x = fix(1) ; x = 1.0
```

4.25 FLIP

In graphics mode 4 and 5, it will toggle the BACKBUFFER bit in the display register of the GBA. In action this means that the screen is flipped

Return value: None

None.	

```
; Set mode 5 and flip screens

graphics 5,0
flip

; draw a line and flip back

line screen,0,0,160,120
flip
```

4.26 FLIPSPRITE sprite,horizontal,vertical

Sets or clears the horizontal and vertical bits of sprite flipping. This causes a "mirror" or "flip" effect when drawing a sprite.

Return value: None.

sprite	A sprite (0-127)
horizontal	0 to reset, anything else sets
vertical	0 to reset, anything else sets

```
; Make a sprite that faces right to face left

flipsprite my_sprite,1,0
```

4.27 FLIPTILE tile,horizontal,vertical

Sets or clears the horizontal and vertical flip bits of a tile. This causes a "mirror" or "flip" effect when drawing a tile.

Return value: None.

tile	A tile address
horizontal	0 to reset, anything else sets
vertical	0 to reset, anything else sets

```
; Flip a tile of an arrow pointing up to make it point down

fliptile tile(4,0,10),0,1
```

4.28 FLOOR (n)

Computes the 16:16 fixed point floor value of n.

The floor of 2.2 is 2.0 and of 2.8 is also 2.0.

Return value: An 16:16 fixed point value (n1/n2).

n	A 16:16 fixed point value

```
; Floor example

x = floor(2.4) ; x = 2.0
x = floor(-2.4) ; x = -2.0
```

4.29 FMUL (n1,n2)

Multiplies the two 16:16 fixed point values n1 and n2.

Return value: An 16:16 fixed point value (n1*n2).

n1	A 16:16 fixed point value
n2	A 16:16 fixed point value

```
; Multiply two fixed-point values

x = fmul(2.5,2.0) ; x = 5.0
```

4.30 FRAME screen,x,y,width,height,color

Draws a color outline of a rectangle on screen starting at (x,y) of width and height.

Return value: None.

screen	Video RAM address
X	The X coordinate to start at
Y	The Y coordinate to start at
width	The width of the rectangle
height	The height of the rectangle
color	A 15-bit color value in BBBBBGGGGGRRRRR format

```
; Draws a green border around the screen

graphics 3,0
frame SCREEN,0,0,240,160,GREEN
```

4.31 FSUB (n1,n2)

Subtracts the two 16:16 fixed point values n1 and n2.

Return value: An 16:16 fixed point value (n1-n2).

n1	A 16:16 fixed point value
n2	A 16:16 fixed point value

```
; Subtract two fixed-point values

x = fsub(10.0,3.5) ; x = 6.5
```

4.32 GETPALENTRY (palette,index,entry)

Gets the 15-bit color value in a 16-color or 256-color palette. To get a color from the 256-color palette, set "index" to 0.

Return value: a 15-bit color value.

palette	SPRITE_PALETTE or BG_PALETTE
index	the 16-color palette index (0-15)
entry	the color entry in the palette

```
; Get the 3rd color in the 6th palette for sprites

clr = getpalentry(SPRITE_PALETTE,5,2)
```

4.33 GRAPHICS mode,sprites

Sets the graphics mode that the GBA is currently in. It will also enable or disable sprites.

Enter a 0 for "sprites" to disable sprites, or any non-zero value to enable them.

Return value: none.

mode	the graphics mode to use (0-5) sprites

```
; Set graphics mode 3 with sprites enabled

graphics 3,true
```

4.34 HIDESPRITE sprite

Sets the position of sprite to somewhere offscreen.

Return value: None.

sprite	A sprite (0-127)

```
; Hide all sprites

for i = 0 to 127
  hidesprite i
next
```

4.35 INPUT mask

Halts execution until the button state for P1 changes for any of the buttons in mask.

Return value: None.

mask	A 16-bit value of buttons that are OR'ed together

```
; Wait for the user to press or release either START or A

input KEY_START or KEY_A
```

4.36 INT (n)

Converts n - a 16:16 fixed point value to an integer.

Return value: An integer.

n	A 16:16 fixed point value

```
; Convert a fixed point value back to integer

y = 10.4
x = int(y) ; x = 10
```

4.37 KEY (mask)

Loads the 16-bit value in the controller register of the GBA and bitwise ANDs it with mask.

Return value: The current state of the buttons in mask. NOTE: there is a 1 for every button that is released, and a 0 for every key that is pressed.

mask	A 16-bit value of buttons that are OR'ed together

```
; Get the current state of the A and B buttons

buttons = key(KEY_A or KEY_B)

; Check the state of each

a = buttons xor KEY_A ; 0=released
b = buttons xor KEY_B ; 0=released
```

4.38 KEYS

Loads the 16-bit value from the P1 controller register.

Return value: The current state of the controller. The register contains a 1 for every button that is not pressed, and a 0 if it is.

None	

```
; Check for a single button pressed - not combinations

select keys xor KEY_ANY
  case KEY_A ; 'A' pressed
  case KEY_B ; 'B' pressed
  case KEY_L ; 'L' pressed
  case KEY_R ; 'R' pressed
  case KEY_UP ; UP pressed
  case KEY_DOWN ; DOWN pressed
  case KEY_RIGHT ; RIGHT pressed
  case KEY_LEFT ; LEFT pressed
 case KEY_START ; START pressed
end select
```

4.39 LINE buffer,x0,y0,x1,y1,color

Draws a Bresenham line of color starting from (x0,y0) to (x1,y1) on the screen buffer in graphics modes 3 or 5.

Return value: None

buffer	Video RAM address
X0	X coordinate of the start pixel
Y0	Y coordinate of the start pixel
X1	X coordinate of the end pixel
Y1	Y coordinate of the end pixel
Color	A 15-bit color value in BBBBBGGGGGRRRRR format

```
; Draw a big X on the screen

graphics 3,0
line SCREEN,0,0,239,159,RED
line SCREEN,239,0,0,159,RED
```

4.40 LOADBYTE

Loads an 8-bit (0-$FF) value from the current data pointer in SRAM. The data pointer can be set with {HYPERLINK "showcommand.php?cmd=RESTORE"} SRAM".

Return value: An 8-bit value (0-$FF).

None.	

```
; Load the first byte in saved-RAM

restore SRAM
b = loadbyte
```

4.41 LOADLONG

Loads an 32-bit (0-$FFFFFFFF) value from the current data pointer in SRAM. The data pointer can be set with "RESTORE SRAM".

Return value: A 32-bit value (0-$FFFFFFFF).

None.	

```
; Load the first 32-bit value stored in SRAM

restore SRAM
x = loadlong
```

4.42 LOADPAL16 palette,index,address

Copies 16 15-bit colors at address to the 16-color palette index offset from palette.

Return value: None

palette	Destination palette (BG_PALETTE or SPRITE_PALETTE)
index	The palette to load (0-15)
address	Source address of the palette in ROM to load

```
; Load a palette into RAM

my_pal: #palette "img.pcx"

start:
  loadpal16 BG_PALETTE,0,my_pal
```

4.43 LOADPAL256 palette,address

Copies 256 15-bit colors at address to palette.

Return value: None

palette	Destination palette (BG_PALETTE or SPRITE_PALETTE)
address	Source address of the palette in ROM to load

```
; Load a 256-color palette into the sprite palette

my_pal: #palette "img.pcx"

start:
  loadpal256 SPRITE_PALETTE,my_pal
```

4.44 LOADSPRITE char,address,blocks

Copies blocks (8x8, 4-bit) pixel data from "address" to the sprite character in VRAM. Note: the sprite data at address must be 1D.

If you are in a bitmapped graphics mode (3-5), then you *must* begin using sprite characters at 512 instead of 0, as VRAM will overlap.

Return value: None.

char	A sprite character (0-1023)
address	Source address in ROM of the sprite data
blocks	Number of 8x8 4-bit blocks of data to copy

```
; Load and make a simple sprite
; sprite.pcx is a 32x32 8-bit sprite

img: #bitmap "sprite.pcx"

start:
  graphics 3,1
  loadsprite 512,img,blocks(32,32,8)
```

4.45 LOADTILES dest,source,blocks

Copies blocks (8x8, 4-bit) of data from "source" to "dest". You can get the proper destination address by using **CHARBLOCK** and **TILEOFFSET**.

Return value: None.

dest	Address of the character base block (destination)
source	Address where the data is located in ROM (source)
blocks	Number of 8x8 4-bit blocks of data to copy

```
; Load a tilemap into RAM into character base block 1
; and offset by 95 tiles.

map_data: ; image is 34x8, 4-bit
  #bitmap "map.png"

start:
  loadtiles
charblock(1)+tileoffset(95),map_data,blocks(34,8,4)
```

4.46 LOADWORD

Loads an 16-bit (0-$FFFF) value from the current data pointer in SRAM. The data pointer can be set with "RESTORE SRAM".

Return value: A 16-bit value (0-$FFFF).

None.	

```
; Load 4 successive, 16-bit values from SRAM

global data(4)

restore SRAM

for i = 0 to 3
  data[i] = loadword
next
```

4.47 MAKEPALETTE palette

Creates a 216-color, universal (web-safe) palette in either BG_PALETTE or SPRITE_PALETTE. You still have access to the other 50 colors available with **GETPALENTRY** and **SETPALENTRY**.

Return value: none.

palette	the destination palette (BG_PALETTE or SPRITE_PALETTE)

```
; Create a 216-color palette for the background tiles

makepalette BG_PALETTE
```

4.48 MAKEROTATION rotation,scalex,scaley,angle

Creates rotation matrix with X scaling factor of scalex, Y scaling factor of scaley and a rotation angle in degrees. Note: a scale factor of 0.5 is double size, and 2 is half size.

Return value: None.

rotation	A rotation matrix (0-3)
scalex	X scaling factor
scaley	Y scaling factor
angle An angle is degrees (0-359)	

```
; Create a rotation matrix that is 45 degrees and 1/2 the
size

makerotation 1,2,2,45

; Make a sprite use that rotation

rotatesprite my_sprite,1
```

4.49 MAKESPRITE sprite,char

Creates a new sprite in 256-color mode and uses the image data for the sprite character char.

Note: MAKESPRITE will cause all the sprite data for "sprite" to be zeroed, clearing out any changes made.

Return value: None.

sprite	A sprite (0-127)
char	A sprite character image (0-1023)

```
; Make a simple sprite

#constant my_sprite 1
#constant sprite_char 678

makesprite my_sprite,sprite_char
```

4.50 MAKETIMER frequency

Creates a new timer to fire "frequency" times every second.

Return value: None.

frequency	Number of times the timer will fire per second (must be > 0)

```
; Create a timer to fire 10 times a second

maketimer 10
starttimer ; start ticking
```

4.51 MAPIMAGE tile,base,width,height

Sets the tiles starting at tile in an area of width and height to incremental values starting at base.

This is used for mapping individual images to a background, when the image is loaded with **LOADTILES** and each tile is uniquely part of the picture.

Return value: None.

tile	A tile address
base	First tile value to write
width	Number of tiles in X direction
height	Number of tiles in Y direction

```
; Load the image of a house (32x32, 4-bit)

#constant house_tile 95

house: #bitmap "house.pcx"

start:
  ; TODO: setup graphics mode and load
  ; "house" to house_tile

  ; Map the image of the house...
  mapimage tile(4,6,7),house_tile,4,4
```

4.52 *MAPTILES tile,address,width,height*

Copies tile data from address to a background at tile in an area bounded by width and height. Same as **BLIT**, but for tiled modes.

Return value: None.

tile	A tile address
Address	Address to copy from
Width	Number of tiles in X direction
Height	Number of tiles in Y direction

```
; Create some map data

my_map:
  data 0,0,0,0
  data 1,0,0,1
  data 0,2,2,0
  data 1,2,2,1

start:
  ; TODO: set graphics mode, etc.

  maptiles tile(0,3,4),my_map,4,4
```

4.53 *MOD*

Returns the modula (remainder) of a calculation.

Return value: An integer.

none	

```
; Calculate the leftover

5 MOD 2 = 1
(5/2 = 2 + 1)
5 can be divided by 2*2.
This results in 5-4 and then there is 1 left over

9 MOD 10 = 9
(9/10 = 0 + 9)
9 can't be divided by 10 in a whole number.
This results in 0 and then there is 9 left over.
```

4.54 MOSAIC bx,by,sx,sy

Pixelates backgrounds and/or sprites by creating a "zoom" effect on the screen. Each sprite and background that wishes to be affected by the effect should have their mosaic bits turned on with either **ENABLEMOSAIC** or **SPRITEMOSAIC**.

Return value: None.

bx	background X scaling factor (0-15)
by	background Y scaling factor (0-15)
sx	sprite X scaling factor (0-15)
sy	sprite Y scaling factor (0-15)

```
; Enable mosaic on background 2 (bitmapped mode)

enablemosaic 2

; Zoom out...

for i = 0 to 15
  mosaic i,i,0,0
next
```

4.55 MOVESPRITE sprite,dx,dy

Adjusts the position of sprite by (dx,dy).

Return value: None.

sprite	A sprite (0-127)
dx	Delta X offset of current position
dy	Delta Y offset of current position

```
; Move a sprite 10 pixels right and 2 pixels up

movesprite my_sprite,10,-2
```

4.56 NOT

A bitwise NOT

Return value: None.

none	

```
; See if d-pad right is pressed.

if not key(key_right)
  ; PRESSED. Do action!
end if
```

4.57 OR

A bitwise OR

Return value: None.

none	

```
; A few examples

    1001  0111  0000
OR  1100  0000  0000
    ----  ----  ----
    1101  0111  0000
```

4.58 ORDERSPRITE sprite,priority

Sets the Z-order priority of sprite. 0 is the highest (top) priority and 3 is the lowest (bottom).

Return value: None.

sprite	A sprite (0-127)
priority	A Z-order priority (0-3)

```
; Order a sprite to appear behind the top most background
; but if front of all others

ordersrpite my_sprite,1
```

4.59 ORDERTILES background,priority

Sets the Z-order drawing priority for background.

Return value: None.

background	A text background (0-3)
prioroity	The priority of drawing (0

```
; Setup background 0 to draw in
; front of background 1

ordertiles 1,1
ordertiles 0,0
```

4.60 PEEK (address)

Loads the 16-bit, halfword value at address.

Return value: The 16-bit value at address.

address	Source address

```
; Load the value of REG_DISPCNT

reg = peek(REG_DISPCNT) ; $4000000
```

4.61 PIXEL (screen,x,y)

Reads the color value of a pixel on screen at (x,y). Note: only available in bitmapped modes.

Return value: The 15-bit color of the pixel at (x,y).

screen	Video RAM address
x	The X coordinate of the pixel
y	The Y coordinate of the pixel

```
; Put random pixels all over the screen

for i = 1 to 200
  plot SCREEN,rnd mod 240,rnd mod 160,rnd
next

; Read the pixel color at 10,12

color = pixel(SCREEN,10,12)
```

4.62 PLAYMUSIC address

Begins to play (and loop) music from address. It will continuously loop until stopped with **STOPMUSIC**.

Return value: None.

address	Address of an imported sound file

```
; Import a WAV file

music: #sound "bg.wav"

; Play it
start:
  playmusic music
```

4.63 PLAYSOUND address

Begins to play a sound from address. It will play over any background music and will stop once completed.

Return value: None.

address	Address of an imported sound file

```
; Import a WAV file

fx: #sound "boom.wav"

start:
  ; play the sound

  playsound fx
```

4.64 PLOT buffer,x,y,color

Sets the color of the pixel at (x,y) on the scree buffer in graphics modes 3 or 5.

Return value: None

buffer	Video RAM address
x	X coordinate of the pixel
y	Y coordinate of the pixel
color	A 15-bit color value in BBBBBGGGGGRRRRR format

```
; Plot a red pixel at 10,10

graphics 3,0
plot SCREEN,10,10,RED
```

4.65 POKE address,n

Stores the 16-bit, halfword value "n" at "address".

Return value: None.

address	Destination address
n	A 16-bit value

```
; Set the graphics mode the old fashioned way

poke REG_DISPCNT,MODE3 or BG2ENABLE
```

4.66 POSITIONSPRITE sprite,x,y

Sets the position of "sprite" to (x,y).

Return value: None.

sprite	A sprite (0-127)
x	X coordinate of the new position
y	Y coordinate of the new position

```
; Move the spaceship sprite to 30,45

positionsprite SHIP,30,45
```

4.67 PRINT address,string

Prints string onto the background tiles at address.

Return value: None.

address	An address gotten with TILE
string	The address of a string stored in ROM

```
; "Hello, world!"
; TODO: load a font and tileset

print tile(0,3,3),"Hello, world!"
```

4.68 RECT screen,x,y,width,height,color

Fills a solid color rectangle on screen starting at (x,y) with width and height.

Return value: None.

screen	Video RAM address
x	The X coordinate to start at
y	The Y coordinate to start at
width	The width of the rectangle
height	The height of the rectangle
color	A 15-bit color value in BBBBBGGGGGRRRRR format

```
; Fill a square green

graphics 3,0
rect SCREEN,0,0,10,10,GREEN
```

4.69 RESETTIMER

Resets the number of fires to 0.

Return value: None.

None.	

```
; Start a new timer

maketimer 1
starttimer

; Wait for it to count to 10 and reset
waittimer 10
resettimer
```

4.70 RGB (red,green,blue)

Creates a 15-bit color value from its separated red, green and blue color components.

Return value: a color.

red	red component (0-31)
green	green component (0-31)
blue	blue component (0-31)

```
; Create a teal color (green/blue)

teal = rgb(0,31,31)
```

4.71 RGBB (color)

Extracts the blue component from color.

Return value: A value from 0-31.

color	A 15-bit color value in the form BBBBBGGGGGRRRRR

```
; Get the blue component of a palette entry

color = getpalentry(BG_PALETTE,0,43)
blue = rgbb(color)
```

4.72 RGBG (color)

Extracts the green component from color.

Return value: A value from 0-31.

color	A 15-bit color value in the form BBBBBGGGGGRRRRR

```
; Get the green component of a palette entry

color = getpalentry(BG_PALETTE,0,43)
blue = rgbg(color)
```

4.73 RGBR (color)

Extracts the red component from color.

Return value: A value from 0-31.

color	A 15-bit color value in the form BBBBBGGGGGRRRRR

```
; Get the red component of a palette entry

color = getpalentry(BG_PALETTE,0,43)
blue = rgbr(color)
```

4.74 RND

Generates a pseudo-random number in the range of 0 to 0x7FFF (RAND_MAX).

Return value: A 15-bit random number.

None.	

```
; Generate 10 random numbers from 0-239

global r(10)

max = (RAND_MAX/240)+1

for i = 0 to 9
  r[i] = rnd/max
next
```

4.75 ROTATEPAL16 palette,index

Rotates all the colors up one entry in a 16-color palette. The last entry is moved to the beginning.

Return value: none.

palette	palette to rotate (BG_PALETTE or SPRITE_PALETTE)
index	the 16-color palette to rotate (0-15)

```
; Rotate a palette 3 times to change a tile

for i = 1 to 3
  rotatepal16 BG_PALETTE,4
next
```

4.76 ROTATEPAL256 palette

Rotates all the colors in a 256-color palette up 1 entry. The last entry is moved back to the beginning.

Return value: none.

palette	the palette to rotate (BG_PALETTE or SPRITE_PALETTE)

```
; Rotate all the entries in the sprite palette

rotatepal256 SPRITE_PALETTE
```

4.77 ROTATESPRITE sprite,rotation

Sets the rotation matrix for sprite to use when rendering to the screen.

Return value: None.

sprite	A sprite (0-127)
rotation	A rotation matrix (0-3) or a negative number to clear rotation

```
; Create a simple matrix and apply it to a sprite

makerotation 0,1,1,45
rotatesprite my_sprite,0

; Wait for the user to press A
; and remove the rotation matrix

input KEY_A
rotatesprite my_sprite,-1
```

4.78 ROUND (n)

Computes the 32-bit integer rounded value of the 16:16 fixed point value n. The rounded value of 2.2 is 2 and of 2.8 is 3.

Return value: A 32-bit integer. (n1/n2).

n	A 16:16 fixed point value

```
; Round example

x = round(1.0) ; x = 1
x = round(2.2) ; x = 2
x = round(-3.8) ; x = -4
```

4.79 SAVEBYTE byte

Writes an 8-bit value (0-$FF) to the data pointer. The data pointer can be initially set with "RESTORE SRAM".

Return value: None.

byte	A single byte value (0-$FF)

```
; Save a byte of data to SRAM

restore SRAM
savebyte $45
```

4.80 SAVELONG long

Writes a 32-bit value (0-$FFFFFFFF) to the data pointer. The data pointer can be initially set with "RESTORE SRAM".

Return value: None.

long	A 4 byte value (0-$FFFFFFFF)

```
; Save a 32-bit value

restore SRAM
savelong $FFEEDDCC
```

4.81 SAVEWORD word

Writes a 16-bit value (0-$FFFF) to the data pointer. The data pointer can be initially set with "RESTORE SRAM".

Return value: None.

word	A 2 byte value (0-$FFFF)

```
; Save a 16-bit value in SRAM

restore SRAM
saveword $FF55
```

4.82 SCANLINE

Returns the current scanline that is being drawn.

Return value: The scanline being drawn. 160 is the scanline that signals a vertical blank.

None.	

```
; Wait for a vertical blank the old fashioned way

while scanline <> 160
  ; Do nothing
loop
```

4.83 SCORE (n)

Converts the integer value "n" to a string. Note: n cannot be a negative value.

Return value: Address to the created string.

n	A 32-bit unsigned integer

```
; A simple message
Print Tile(8,1,1),"Seconds Elapsed:"

; Create a timer to fire once every second
maketimer 1
starttimer

; Loop until the user presses A
while Key(KEY_A)
  print tile(8,18,1),score(clocktimer)
loop
```

4.84 SCREEN

Return the current back buffer address or $6000000 if not in modes

4 or 5

Return value: Address to the current back buffer.

none	A 32-bit unsigned integer

```
; Set mode 3 and display a screen

        Graphics 3, TRUE
        wallpaper SCREEN, splash
```

4.85 SCREENBLOCK (n)

Calculates the address of screen base block n.

Return value: A 32-bit address.

n	A screen base block from 0-31

```
; Import some data

my_data: #import "bin_data.bin"

start:
  ; Copy the data to a screenblock
  copy screenblock(3),my_data,128
```

4.86 SCROLL background,x,y

Scrolls "background" by (x,y) pixels. This sets the scroll value, and does not adjust the current scroll settings.

Return value: none.

background	the background layer to scroll (0-3)
X	the number of horizontal pixels to scroll
Y	the number of vertical pixels to scroll

```
; TODO: get graphics mode
; TODO: enable background 0 with a map

; Loop forever, scrolling the background
; horizontally...

x = 0

while
  x = x + 1
  vblank
  scroll 0,x,0
loop
```

4.87 SEED n

Sets the current random number seed to "n".

Return value: None.

n	Any 32-bit number

```
; Loop until the user presses START

s = 0

while key(KEY_START)
  s = s + 1
loop

; Set a random seed based on the user

seed s
```

4.88 SETPALENTRY palette,index,entry,color

Sets the color of an entry in either a 16-color or 256-color palette. To set a 256-color palette entry, set "index" to 0.

Return value: none.

palette	BG_PALETTE or SPRITE_PALETTE
index	a 16-color palette to edit (0-15)
entry	a color entry from the palette
color	a 15-bit color value in BBBBBGGGGGRRRRR format

```
; Get a palette entry color

color = getpalentry(SPRITE_PALETTE,2,3)

; Get the individual components

r = rgbr(color)
g = rgbg(color)
b = rgbb(color)

; Swap the red and blue components and set it

setpalentry SPRITE_PALETTE,2,3,rgb(b,g,r)
```

4.89 SIN (degrees)

Computes the sine of an angle in degrees between 0 and 359.

Return value: A fixed-point value that is the sine of degrees.

degrees	An integer angle between 0 and 359

```
; Compute the sine of 45

x = sin(45) ; x = 0.707
```

4.90 SIZESPRITE sprite,shape,size

Sets the size of sprite based on constant shape and size parameters.

	Square	Wide	Tall
SIZE_8	8x8	16x8	8x16
SIZE_16	16x16	32x8	8x32
SIZE_32	32x32	32x16	16x32
SIZE_64	64x64	64x32	32x64

Return value: None.

sprite	A sprite (0-127)
shape	SQUARE, WIDE, TALL or DOUBLE
size	SIZE_8, SIZE_16, SIZE_32 or SIZE_64

```
; Make a sprite and size it to 16x32

makesprite my_sprite,3
sizesprite my_sprite,TALL,SIZE_32
```

4.91 (n) SL number

Bit shift left. Shift bits of the *number* n-places to the left

Return value: The address of sprite n.

n	Number of steps to shift

```
; Multiply a number by 6 the fast way

Value = 3 SL 1
; The result will be 6
```

4.92 SPRITE (n)

Calculates the base address in RAM sprite "n". Note: this address is *not* OAM.

Return value: The address of sprite n.

n	A sprite (0-127)

```
; Get the x position of a sprite

addr = sprite(my_sprite) + 2
x = peek(addr) and $1FF
```

4.93 SPRITEFRAME (sprite)

Gets the current frame block of animation that sprite is using.

Return value: The current sprite character of sprite.

sprite	A sprite (0-127)

```
; Animate a sprite until reaching a certain block

while spriteframe(my_sprite) <> 128
  animsprite my_sprite,64,128,8
loop
```

4.94 SPRITEMOSAIC sprite,enable

Toggles on or off the bit in sprite RAM that tells the GBA to let the sprite mosaic effect this sprite.

Return value: none.

sprite	the sprite to effect (0-127)
enable	zero to turn off mosaic bit, non-zero to enable

```
; Enable a sprite mosaic and zoom it

spritemosaic my_sprite,1

for i = 0 to 15
  mosaic 0,0,i,i
next
```

4.95 SPRITEX (sprite)

Gets the current X coordinate of sprite. Note: sprite coordinates range from (-272,-96) to (239,159) and will wrap around as needed.

Return value: The X coordinate of sprite.

sprite	A sprite (0-127)

```
; SPRITEX Example code.

; ToDo setting up the sprites

#CONSTANT Mario 2
Xposition = SpriteX(Mario)
```

4.96 SPRITEY (sprite)

Gets the current Y coordinate of sprite. Note: sprite coordinates range from (-272,-96) to (239,159) and will wrap around as needed.

Return value: The Y coordinate of sprite.

sprite	A sprite (0-127)

```
; SPRITEY Example code.

; ToDo setting up the sprites

#CONSTANT Mario 2
Yposition = SpriteY(Mario)
```

4.97 (n) SR number

Bit shift right. Shift bits of the *number* n-places to the right.

Return value: The address of sprite n.

n	Number of steps to shift

```
; Divide a number by 6 the fast way

Value = 3 SR 1
; The result will be 1
```

4.98 STARTTIMER

Begins the timer firing.

Return value: None.

None.	

```
; Timer Example Code
;
#include "gba.dbc"
; Create a timer to fire once every second
; and get it ticking

MakeTimer 1
StartTimer

; Loop until the user presses A
while Key(KEY_A)
; Wait for a vertical blank
Vblank
; Display the number of seconds elapsed
Print Tile(8,18,1),Score(ClockTimer)
Loop
; Reset and stop the timer
ResetTimer
StopTimer
End
```

4.99 STOPMUSIC

Turns off any music that is playing in the background.

Return value: None

None.	

```
; Sound/Music Example Code
;
; Include constants
#include "gba.dbc"
; Import some background music
music:
#sound "theme.wav"
; Import a sound byte
coin:
#sound "coin.wav"
; Start the program
start:
; Begin playing background music
PlayMusic music
while
; Every time the user hits A, play a sound
if Key(KEY_A) = 0 then PlaySound coin

; If the user presses START then stop music
if Key(KEY_START) = 0 then StopMusic
loop
```

4.100 STOPSOUND

Stops any sound that is currently playing.

Return value: None.

None.	

4.101 STOPTIMER

Stops the timer from firing.

Return value: None.

None.	

```
; Make and start a timer

MakeTimer 1
StartTimer

; Wait for 10 seconds

WaitTimer 10

; Stop the timer

StopTimer
```

4.102 TAN (degrees)

Computes the tangent of an angle in degrees between 0 and 359.

Return value: A fixed-point value that is the tangent of degrees.

degrees	An integer angle between 0 and 359

```
; Get the tangent of 45 degrees

x = tan(45) ; x = 1.0
```

4.103 TILE (block,x,y)

Calculates the address of the tile at (x,y) of the screen base block, block.

Return value: Address of the tile.

block	A screen base block (0-31)
x	X coordinate of the tile
y	Y coordinate of the tile

```
; Print text at a particular tile

Print Tile(8,0,2),"Hello, world!"

; Flip the 'e' tile vertically

FlipTile Tile(8,1,2),0,1
```

4.104 TILEOFFSET (blocks)

Calculates the number of bytes that blocks of 8x8 16-color blocks uses in RAM.

Return value: Number of bytes needed.

blocks	Number of 8x8 16-color tiles

```
; Load some tiles

font: #bitmap "font.png"
more_data: #bitmap "house.png"

start:
  LoadTiles Charblock(0),font,95

  ; Load some more data after the font

  LoadTiles Charblock(0)+TileOffset(95),more_data,4
```

4.105 TRIANGLE buffer,x0,y0,x1,y1,x2,y2,color

Fills a triangle of color starting from (x0,y0) to (x1,y1) and (x2,y2) on the screen buffer in graphics modes 3 or 5.

Return value: None

buffer	Video RAM address
x0	X coordinate of the first pixel
y0	Y coordinate of the first pixel
x1	X coordinate of the second pixel
y1	Y coordinate of the second pixel
x2	X coordinate of the last pixel
y2	Y coordinate of the last pixel
color	A 15-bit color value in BBBBBGGGGGRRRRR format

```
; Draw a large triangle on the screen

Graphics 3,0
Triangle SCREEN,0,159,120,0,239,159,RED
```

4.106 UPDATESPRITES

Copies all modified sprite data in RAM to OAM.

Return value: None.

None.	

```
; Main game loop

while

  ; Wait for vertical blank and update

  vblank : UpdateSprites

  ; TODO: game stuff
loop
```

4.107 VBLANK

Halts execution until a vertical blank occurs.

Return value: None.

None.	

```
; Wait for a vertical blank

vblank

; TODO: draw stuff during blank
```

4.108 WAITTIMER count

Halt the program and wait until the fire counter is greater than or equal to count.

Return value: None.

count	The number of fires to wait for

```
; Make and start a timer

MakeTimer 1
StartTimer

; Wait for 10 seconds

WaitTimer 10

; Stop the timer

StopTimer
```

4.109 WALLPAPER buffer,address,wait

Uses DMA to copy (fast) the stored image at address to the screen buffer in graphics modes 3 or 5. Note: In mode 3, the image must be 16-bit and 240x160. In mode 5 is must be 16-bit and 160x120.

Return value: None

buffer	Video RAM address
address	The address of an imported picture file
wait	Non-zero if the program should halt until the image is pasted

```
; Render a picture to the screen
; A 240x160, 24-bit image

pic: #bitmap "my_pic.pcx"
start:
  Graphics 3,0
  Wallpaper SCREEN,pic,1
```

4.110 XOR

A exclusive bitwise OR

Return value: Integer

none	

```
; XOR Example Code
; 1 or the other, but not both

%1011 xor %0110 = %1101

     1001  0111  0000
XOR  1100  0000  0000
     ----  ----  ----
     0101  0111  0000
```

5 The commandset – Sorted by area

Commands can also be ordered according their purpose. This chapter will give and overview of each command and in which area it can be used.

5.1 Background and Tiles

Command	Description
BLOCKS	Computes the number of 8x8, 4-bit blocks of data an image takes up.
CHARBLOCK	Calculates the address of character base block.
CLEARTILES	Erases background tile data.
COLORTILE	Changes the palette used by the tile at address.
DISABLEMOSAIC	Turns off the mosaic bit for a background.
DISABLETILES	Disables a background layer.
ENABLEMOSAIC	Turns on the mosaic bit for a background.
ENABLETILES	Enables a background layer.
FLIPTILE	Sets or clears the horizontal and vertical flip bits of a tile.
LOADTILES	Copies blocks (8x8, 4-bit) of data from "source" to "dest".
MAPIMAGE	Sets an area of tiles starting at tile
MAPTILES	Copies tile data from address to a background.
ORDERTILES	Sets the Z-order drawing priority for background
PRINT	Prints string onto the background tiles.
SCREENBLOCK	Calculates the address of screen base block "n".
SCROLL	Scroll a background
TILE	Calculates the address of the tile of the screen.
TILEOFFSET	Calculates bytes that tiles uses.

5.2 Bitmap graphics

Command	Description
BLIT	Pastes bitmap image to screen.
CIRCLE	Draws a circle.
CLS	Erases a screen buffer.
FLIP	Toggle the BACKBUFFER bit in the display register of the GBA.
FRAME	Draws a rectangle outline.
GRAPHICS	Set the graphics mode
LINE	Draws a solid line
MOSAIC	Set the parameters for the GBA' s mosaic effect
PIXEL	Reads the color value of a pixel
PLOT	Sets the color of the pixel
RECT	Fills a solid color rectangle on screen
RGBG	Extracts the green component from color.
RGBR	Extracts the red component from color.
RGBB	Extracts the blue component from color.
RGB	Creates a 15-bit color value from its separated red, green and blue color components.
SCANLINE	Determines the currently rendering scanline
SCREEN	Return the current back buffer address
TRIANGLE	Fills a triangle of color.
VBLANK	Halts execution until a vertical blank occurs.
WALLPAPER	Copy image to screen buffer in graphics modes 3 or 5.

5.3 Compiler directives

Command	Description
#ALIGN	Aligns the ROM binary along a bytes boundary.
#BITMAP	Extracts the pixel or tile data from an image file.
#CONSTANT	Creates a new constant identifier.
#FONT	Sets the current "lookup" font table to string.
#IMPORT	Imports a binary file.
#INCLUDE	Includes source code into your program.
#PALETTE	Extracts palette information from an image file.
#POOL	Set registers with values above 255.
#SOUND	Extracts and converts sound from a file and compiles it into the binary.
#TITLE	Sets the 12-character name of your game in the compiled ROM header.

5.4 Extended Basic functions

Command	Description
ABS	Computes the absolute value of an integer or fixed-point value.
ALSO	A logical AND
AND	A bitwise AND
COPY	Copies words of data from source to dest address.
COS	Computes the cosine of an angle in degrees.
ERASE	Zeroes words (32-bit) of data at address.
FADD	Adds the two 16:16 fixed point values.
FDIV	Divides the two 16:16 fixed point values.
FIX	Converts a 32-bit integer to a 16:16 fixed point number.
FLOOR	Computes the 16:16 fixed point floor value.
FMUL	Multiplies the two 16:16 values.
FSUB	Subtracts the two 16:16 values n1 and n2.
INT	Converts a 16:16 value to an integer.
MOD	Returns the modula (remainder)
NOT	A bitwise NOT
OR	A bitwise OR
PEEK	Loads the 16-bit, halfword value at address.
POKE	Stores a halfword value at an address
RND	Generates a pseudo-random number
ROUND	Computes the 32-bit integer rounded value of the 16:16 fixed point value.
SCORE	Convert an integer to a string
SEED	Sets the random number seed
SIN	Computes the sine of an angle.
SL	Bit shift left
SR	Bit shift right
TAN	Computes the tangent of an angle.
XOR	Exclusive bitwise OR

5.5 Input functions

Command	Description
{ HYPERLINK "showcommand.php?id=14" }	Halts execution until the button state changes for buttons in mask.
{ HYPERLINK	Loads the 16-bit value in the controller register of the GBA and bitwise ANDs it

"showcommand.php?id=15"}	with mask.
{ HYPERLINK "showcommand.php?id=13"}	Loads the value from the controller register.

5.6 Palette functions

Command	Description
GETPALENTRY	Retrieve a color from a palette
{ HYPERLINK "showcommand.php?id=70" }	Copies 16 15-bit colors at address to the 16-color palette index offset from palette.
LOADPAL256	Copies 256 15-bit colors at address to palette.
MAKEPALETTE	Create a 216-color, universal palette
RGB	Create 15-bit color from separate components
RGBG	Extracts the green component from color.
RGBR	Extracts the red component from color.
RGBB	Extracts the blue component from color.
ROTATEPAL16	Rotates all of the colors in a 16-color palette
ROTATEPAL256	Rotates all the colors in a 256-color palette
SETPALENTRY	Set the color of a palette index entry

5.7 Sound functions

Command	Description
PLAYMUSIC	Begins to play (and loop) music
PLAYSOUND	Begins to play a sound.
STOPMUSIC	Turns off music playing.
STOPSOUND	Stops any sound playing.

5.8 Sprite functions

Command	Description
ANIMSPRITE	Sets the current animation frame for sprite.
BUMPSPRITES	Checks to see if two sprites are overlapping.
COLORSPRITE	Sets the sprite to 16-color mode and selects the palette index to use.
FLIPSPRITE	Sets or clears the horizontal and vertical bits of sprite flipping.
HIDESPRITE	Sets the position of sprite to somewhere offscreen.
LOADSPRITE	Load a sprite image into sprite RAM.
MAKEROTATION	Creates rotation matrix for sprites.
MAKESPRITE	Creates a new sprite
MOVESPRITE	Adjusts the position of sprite
ORDERSPRITE	Sets the Z-order priority of sprite
POSITIONSPRITE	Sets the position of a sprite
ROTATESPRITE	Sets the rotation matrix for sprite
SIZESPRITE	Sets the size of sprite.
SPRITE	Calculates address of sprite n.
SPRITEFRAME	Gets the current frame block of sprite animation
SPRITEMOSAIC	Toggles the mosaic bit of a sprite
SPRITEX	Gets X coordinate of sprite.
SPRITEY	Gets Y coordinate of sprite.
UPDATESPRITES	Copies all modified sprite data in RAM to OAM.

5.9 SRAM functions

Command	Description
LOADBYTE	Loads an 8-bit (0-$FF) value from SRAM
LOADLONG	Loads an 32-bit value from the current data pointer in SRAM.
LOADWORD	Loads an 16-bit value from data pointer in SRAM.
SAVEBYTE	Writes an 8-bit value to the data pointer in SRAM
SAVELONG	Writes a 32-bit value to the data pointer in SRAM
SAVEWORD	Writes a 16-bit value to the data pointer in SRAM

6 The commandset – Program flow

The following commands are use for program flow, decisions and loops.

6.1 *FOR - NEXT*

A FOR NEXT loop can be used to go through a pre-defined set of stages. Counting will by default go into steps of 1.

Related keywords:

TO	Sets the range of the loop
STEP	Reach the counter with x (Default 1)
DOWNTO	Counting backwards (Default -1)

```
; FOR NEXT loop Example Code

; Counts 1,3,5,7,9 and then hops out of the loop
FOR Count = 1 TO 10 STEP 2
  Label[Count] = Count
NEXT

; Counts 4,3,2,1 and then hops out of the loop
FOR Count = 4 DOWNTO 1
  Label[Count] = Count
NEXT
```

6.2 *FUNCTION functionname*

Define a function. A function is a piece of self containt code. Which can return a value or just preform output. A function must have a unique name

Related keywords:

END FUNCTION	Mark the end of the function
RETURN	Return the function with a result

```
; FUNCTION Example Code

global x[10]

function fill_array(size)

    ; Fill the array with data
    for i = 0 to size - 1
        read x[i]
    next

    ; Return the last value
    return x[size - 1]
end function
```

6.3 GOSUB label

Jump to a sub routine. A sub routine can be a piece of code that does a specific task. A GOSUB always points to a label

Related keywords:

RETURN	Return from the sub to the line next to the GOSUB call.

```
; GOSUB Example Code

GOSUB Fill_Array

; Loop endless
WHILE
LOOP

Fill_Array:
    ; Fill the array with data
    for i = 0 to size - 1
        read x[i]
    next
RETURN
```

6.4 GOTO label

Jumps to a specific label. The use of GOTO should be avoided as much as possible.

Related keywords:

None	

```
; GOTO Example Code

GOTO Fill_Array

Lable2:

; Loop endless
WHILE
LOOP

Fill_Array:
    ; Fill the array with data
    for i = 0 to size - 1
        read x[i]
    next
    GOTO Label2
```

6.5 IF <condition>

With IF you can make conditional jumps and complex descision structures

Related keywords:

ELSE	Do the next block if condition is FALSE
END IF	End the conditonal check block
THEN	When If is use as a single like the keyword THEN must be used

```
; IF THEN ELSE Example Code

; Conditon check and what the do on one line
IF KEY(KEY_R) = 0 THEN GOTO Fill_Array

Lable2:

IF X[1] = 0
    X[1] = x[1] + 1
  ELSE
    X[1] = x[1] + 2
END IF

; Loop endless
WHILE
LOOP

Fill_Array:
    ; Fill the array with data
    for i = 0 to size - 1
        read x[I]
    next
    GOTO Label2
```

6.6 SELECT <value>

If a variable can have multiple calue then it may be better to use the SELECT statement instead of constructing a large IF THEn ELSE tree

Related keywords:

CASE	Check for a value
END SELECT	End the conditonal check block

```
; SELECT CASE Example Code

; Get a random number from 0 to 2
Value = RND MOD 3

SELECT Value
  CASE 0
    ; Value = 0
  CASE 1
   ; Value = 2
  CASE 2
    ; Value = 2
END SELECT
```

6.7 WHILE <condition>

Loop until the condition meets its criteria

Related keywords:

LOOP	Mark the end of the WHILE block

```
; SELECT CASE Example Code

; Get a random number from 0 to 200
Value = RND MOD 201

WHILE Value <> 13
  ; Do this because the value isn’t 13
  Value = RND MOD 201
LOOP

; If you arrive here then value = 13
```

7 The commandset – Other related

The following commands couldn't be placed in any other group.

7.1 Calculate

Tools to do basic calculations and comparisons.

Related symbols:

+	Add
-	Substract
*	Multiply
\	Divide
<	Smaller then
>	Larger then
<=	Smaller or equal
=>	Larger or equal
<>	Not equal to
FALSE	Boolean NO
TRUE	Boolean YES

7.2 Data sets

Store and retrieve data.

Related keyword:

DATA	Hold a data element
READ	Reads a data element
RESTORE	Re-position the data pointer

```
; DATA Example Code

GLOBAL Y(10)

Block1:
DATA 7, 1, 2, 3, 4, 5, 6, 7

Block2:
DATA 9, 8, 7, 6, 5

; Read the first 4 elements of data block2
RESTORE Block2
FOR T = 0 TO 3
  READ X
  Y[T] = X
NEXT

; Next read all elements of Block1
RESTORE Block1
READ NumbOfElements
FOR T = 1 TO NumbOfElements
  READ X
  Y[T] = X
NEXT
```

7.3 Variable storage

A variable can be defined locally, such as within a function or globally and accessible throughout the whole project

Related keyword:

GLOBAL	Data accessible throughout the whole project
LOCAL	Only local available

l

8 Appendix

Nothing yet

Filename:	DragonBasic manual 2.doc
Directory:	D:\Data\Word\Cursussen
Template:	C:\Program Files\Microsoft Office\Templates\Normal.dot
Title:	Flextrans vertaalsite
Subject:	
Author:	van Zoelen A.A.
Keywords:	Flextrans vertaalsite
Comments:	
Creation Date:	12/06/03 4:19 PM
Change Number:	27
Last Saved On:	17/06/03 12:16 PM
Last Saved By:	Card Boardgames
Total Editing Time:	476 Minutes
Last Printed On:	17/06/03 12:17 PM

As of Last Complete Printing

Number of Pages:	57
Number of Words:	9,693 (approx.)
Number of Characters:	55,252 (approx.)