

User's Manual



Enhanced Creations

DirectQB

User's Manual

DirectQB

A Game Programming Library for QuickBasic 4.5

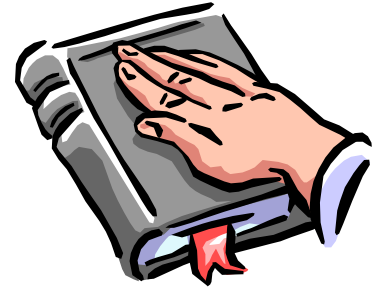
Version 1.61

Released January 6, 1999

By Angelo Mottola – Enhanced Creations 1998-99

Edited by Jon Petrosky (Plasma357)

Legal Disclaimers and Copyrights



Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. This manual is a third-party document is based on the original DirectQB text-only manual and is not supported or endorsed in any way by Angelo Mottola or Enhanced Creations.

DirectQB © 1999 Enhanced Creations. All rights reserved.

Microsoft, MS, MS-DOS, Windows, QuickBasic, and QBasic are either registered trademarks or trademarks of Microsoft Corporation in the USA and other countries.

IBM is a registered trademark of International Business Machines Corporation.
PC Paintbrush is a registered trademark of WordStar Atlanta Technology Center.

Limitation of Liability

THE DIRECTQB SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL THE PUBLISHER, ENHANCED CREATIONS, OR THE PUBLISHER'S OR ENHANCED CREATION'S SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ENHANCED CREATIONS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

High Risk Activities

DirectQB is not fault-tolerant and is not designed, manufactured, or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). The publisher, Enhanced Creations, and their suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

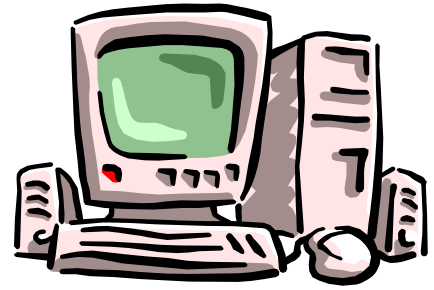
Freeware Concept

DirectQB follow the general rules of the "freeware" concept: It may be legally copied and distributed without limitation, so long as all the original files are included in their unmodified form, and it is not sold.

DirectQB Usage Agreement

If you use DirectQB in your program, you must include both "DirectQB" and "Angelo Mottola" in your program credits. If you would like to modify DirectQB for any purpose other than internal or private use, you must contact Angelo Mottola first.

Contents



Legal Disclaimers and Copyrights 3

Contents 4

Document Conventions 9

Programming Style in This Manual 9

New Features 11

Chapter 1 Introduction 14

Installing DirectQB 14

Features 17

Included Files 19

Accessing the Library 19

Basic Concepts 20

Library Limitations 23

Chapter 2 Function Reference 24

DQBangle Function 24

DQBasc Function 25

DQBbgtri Sub 26

DQBbox Sub 27

DQBboxf Sub 28

DQBbPut Sub 28

DQBbtri Sub 30

DQBbttri Sub 30

DQBchDir Function 31

DQBclearLayer Sub 32

DQBclose Sub 32

DQBcloseDataFile Sub 33

DQBcloseFLI Sub 33

DQBcollide Function 34

DQBcollideOnLayer Function 34

DQBcopyBlendLayer Sub 35

DQBcopyHitLayer Sub 35

DQBcopyLayer Sub 36

DQBcopyTransLayer Sub 37

DQBcreateBMap Function	37
DQBdir\$ Function	38
DQBdrive\$ Function	40
DQBellipse Sub	40
DQBerror\$ Function	41
DQBfadeIn Sub	42
DQBfadeStepIn Sub	42
DQBfadeStepTo Sub	43
DQBfadeTo Sub	43
DQBfilterBox Sub	44
DQBfindCol Function	45
DQBfindPalCol Function	45
DQBfPut Sub	46
DQBframeReady Function	46
DQBfttri Sub	47
DQBget Sub	48
DQBgetBMap Function	49
DQBgetCol Sub	50
DQBgetPal Sub	51
DQBgline Sub	51
DQBgtri Sub	52
DQBhPut Sub	53
DQBid\$ Function	53
DQBinit Function	54
DQBinitText Sub	55
DQBinitVGA Sub	55
DQBinkey\$ Function	56
DQBinstallKeyboard Sub	57
DQBinstallSB Function	57
DQBinUse Function	58
DQBjoyDetected Function	59
DQBjoyFire Function	60
DQBjoyMove Function	60
DQBjoyX Function	62
DQBjoyY Function	62
DQBkey Function	63
DQBlen Function	64
DQBline Sub	65
DQBloadBMap Function	65
DQBloadFont Function	66
DQBloadImage Function	66
DQBloadRawSound Function	67
DQBloadSound Function	68
DQBmapLayer Function	69
DQBmouseDetected Function	69
DQBmouseHide Sub	70
DQBmouseLB Function	71

DQBmouseRB	Function	71
DQBmouseShow	Sub	72
DQBmouseX	Function	72
DQBmouseY	Function	72
DQBmPut	Sub	73
DQBnumDrives	Function	73
DQBopenDataFile	Function	74
DQBopenFLI	Function	75
DQBpaint	Sub	76
DQBpalOff	Sub	77
DQBpalRotate	Sub	77
DQBpath\$	Function	78
DQBpauseSound	Sub	78
DQBpeek	Sub	78
DQBplayFLI	Function	79
DQBplayFLIstep	Sub	81
DQBplaySound	Sub	82
DQBpoint	Function	84
DQBpoke	Sub	84
DQBpollJoy	Sub	86
DQBpPut	Sub	86
DQBprint	Sub	87
DQBprints	Sub	88
DQBpset	Sub	89
DQBput	Sub	89
DQBputOver	Sub	91
DQBreadBit	Function	92
DQBreadKey	Function	92
DQBremoveBMap	Sub	92
DQBremoveKeyboard	Sub	93
DQBremoveSB	Sub	93
DQBresetBit	Function	94
DQBresetJoy	Sub	94
DQBresetMouse	Sub	94
DQBresumeSound	Sub	95
DQBrPut	Sub	95
DQBsaveBMap	Function	96
DQBsaveImage	Function	97
DQBscroll	Sub	98
DQBscrollArea	Sub	99
DQBsetBaseLayer	Function	99
DQBsetBIOSfont	Sub	100
DQBsetBit	Function	101
DQBsetBMap	Sub	101
DQBsetClipBox	Sub	101
DQBsetCol	Sub	102
DQBsetCollideMethod	Sub	102

DQBsetDrive	Sub	103
DQBsetFont	Sub	103
DQBsetFontTexture	Sub	104
DQBsetFrameRate	Sub	105
DQBsetJoySens	Sub	105
DQBsetMousePos	Sub	106
DQBsetMouseRange	Sub	106
DQBsetMouseShape	Sub	106
DQBsetMouseSpeed	Sub	107
DQBsetPal	Sub	107
DQBsetSolidPut	Sub	108
DQBsetTextBackCol	Sub	108
DQBsetTextBMap	Sub	109
DQBsetTextSpacing	Sub	109
DQBsetTextStyle	Sub	109
DQBsetTextureSize	Sub	110
DQBsetTransPut	Sub	111
DQBsetVoiceVol	Sub	111
DQBsetVolume	Sub	112
DQBshiftLeft	Function	112
DQBshiftRight	Function	112
DQBsize	Function	113
DQBsort	Sub	114
DQBspPut	Sub	114
DQBstopVoice	Sub	115
DQBtoggleBit	Function	115
DQBtPut	Sub	116
DQBtri	Sub	116
DQBttri	Sub	118
DQBunpackBMap	Function	119
DQBunpackCursor	Function	120
DQBunpackFont	Function	120
DQBunpackImage	Function	121
DQBunpackPal	Function	121
DQBunpackSound	Function	122
DQBunpackSprite	Function	123
DQBunpackUser	Function	123
DQBver	Function	124
DQBwait	Sub	125
DQBwaitKey	Sub	125
DQBxPut	Sub	125

Appendix A Library Constants 128

Appendix B Keyboard Scan Codes 130

Appendix C Library File Formats 131

Palette Data 131

Mouse Cursor 131

Font Data 133

Blender Map Data 134

Datafile File Format 134

BSAVE File Format 137

Additional File Formats 138

Appendix D Known Bugs 139**Appendix E Version History 140****Appendix F Inside Library Modules 143****Appendix G Error Messages 145****Credits and Final Words 147**

Document Conventions



This manual uses the following typographic conventions.

Example of convention	Description
SUB, IF, CASE ELSE, PRINT	Words in bold indicate language-specific keywords.
<i>setup</i>	Words you're instructed to type appear in bold and italics.
<i>Event-driven</i>	In text, italic letters indicate defined terms, usually the first time they occur in the manual. Italic formatting is also used occasionally for emphasis.
<i>Variable</i>	In syntax, italic letters indicate placeholders for information you supply.
<code>PRINT "Hello, world!"</code>	This font is used for code.
<code>QB.EXE</code>	Words in all capital letters indicate filenames.
<code>DOWN ARROW</code>	Individual direction keys are referred to by the direction arrow on the key top (LEFT, RIGHT, UP, or DOWN). The phrase "arrow keys" is used when describing these keys collectively.
<code>BACKSPACE, HOME</code>	Other navigational keys are referred to by their specific names.
Expanded Memory Services (EMS)	Acronyms are usually spelled out the first time they are used.

Programming Style in This Manual

The following guidelines are used in the examples in this manual.

- QuickBasic keywords appear with all letters capitalized:
 ' SUB, IF, CHDIR, PRINT, and FOR are keywords.
 PRINT "DirectQB"
- Line numbers are not used.

-
- An apostrophe (') introduces comments, instead of **REM**:

```
' This is a comment: these two lines  
' are ignored when the program is running.
```

- Control-flow blocks are indented two spaces:

```
FOR x = 0 to 9  
  PRINT x  
  PRINT x * 25  
NEXT
```

- Lines too long to fit on one line in the manual may be continued on the next line using a line-continuation character, which is a single leading space followed by an underscore (_):

```
DECLARE SUB DQBcollide (BYVAL x1, BYVAL y1, BYVAL Seg1, BYVAL _  
Off1, BYVAL x2, BYVAL y2, BYVAL Seg2, BYVAL Off2)
```

- Constant names use all capital letters:

```
TRUE  
FALSE  
TEXTURED  
RIGHT
```

New Features



The following is a brief description of the new and improved library features for DirectQB version 1.61.

- EMS memory can now be used to store your own custom data.
- Sound output quality has improved a lot (no more "clicks"!) You can now play sounds at 23,000 Hz without problems, which is almost the same quality of "Bells, Whistles, and Sound Boards!" (BWSB)
- **DQBemsSeg** has been replaced by **DQBmapLayer**, which is more versatile for advanced memory handling.
- The blender map system has been completely rewritten. It now allows up to 10 maps at the same time, with smarter memory management. (64K base memory is no longer required for each map!)
- **DQBloadLayer** and **DQBsaveLayer** have been replaced by **DQBloadImage** and **DQBsaveImage**, which allow loading and saving images of any size, using BSV, BMP, or PCX formats.
- Text output routines have been improved. All the text style combinations work at the same time, and some other bugs have been fixed.
- All the library functions have been accelerated a little.
- Texture mapping now supports bilinear filtering, but you need a gradient palette for it to work correctly.
- **DQBcopyBlendLayer** and **DQBcopyHitLayer** have been added. The first blends one layer onto another one, and the second creates a collision layer to be used with the **DQBcollideOnLayer** function.
- Datafile support has been added! DirectQB now has a datafile decoding module, plus an encoder program that allows you to create compressed datafiles that are encrypted by a user-specified password. A datafile can store images, sprites, sounds, palettes, fonts, mouse cursors, and any kind of user defined data. Your program can decode this data on the fly (if you know the correct password, of course!)
- **DQBwaitFrame** Sub has been replaced by **DQBframeReady** Function, which returns true if enough time has passed to synchronize the graphics output to the frame rate, otherwise false.
- **DQBgline** draws a line interpolating colors between the two indexes specified at the endpoints.

- The default font now supports extended characters (ASCII 128-255).
- Directory handling routines now work with Windows long filenames (if running under Windows).
- DQB Library Manager now asks where your copy of QuickBasic 4.5 is first, and scans the drive for it only if you tell it to do so.
- DQB Tools has been modified to display extended characters and long filenames.

Modified Functions

The following is a list of new or modified functions.

DQBbPut	DQBinit	DQBsaveBMP
DQBchDir	DQBinstallSB	DQBsaveImage
DQBclose	DQBloadBMap	DQBsaveLayer
DQBcloseDataFile	DQBloadImage	DQBsetBIOSfont
DQBcopyBlendLayer	DQBloadLayer	DQBsetBMap
DQBcopyHitLayer	DQBmapLayer	DQBsetTextBMap
DQBcreateBMap	DQBopenDataFile	DQBunpackBMap
DQBdir\$	DQBpalRotate	DQBunpackCursor
DQBemsSeg	DQBpath\$	DQBunpackFont
DQBfilterBox	DQBpeek	DQBunpackImage
DQBframeReady	DQBplaySound	DQBunpackPal
DQBfttri	DQBpoke	DQBunpackSound
DQBgetBMap	DQBpPut	DQBunpackSprite
DQBgline	DQBprint	DQBunpackUser
DQBid\$	DQBremoveBMap	DQBwaitFrame

See Chapter 2 for details on each function. Also, as the library still has limitations, be sure to take a look at Chapter 1, “Library Limitations”.

This will probably be the last major release of the DirectQB library. Other minor versions may come out, but they will be almost entirely about bug fixes. I still think a matrix calculation module would be useful, so maybe I'll create it someday. The same goes for a possible modem-handling module: this would allow multiplayer support, but only in direct connections, not using the Internet (no TCP/IP support). For now, use this version for your programs, as it's stable and will be around for a while. Have fun with DirectQB, and remember to visit the Enhanced Creations website at:

<http://ec.neozones.com>

(See note on next page)

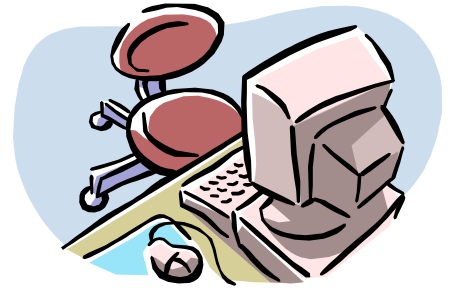
Editor's Note Enhanced Creations shut down January 6, 2000 and has stopped all DirectQB developments. DirectQB 1.61 is the last official version and no future updates should be expected. They recently restarted in February 2001 as Enhanced Creations++. They no longer develop QuickBasic programs, although you can still download their old QuickBasic products from the new Enhanced Creations++ site at:

<http://www.ecplusplus.com>

Additionally, Excalibur North provides recent information and bug reports for DirectQB because Enhanced Creations++ no longer does. You can visit the DirectQB section on their site at:

<http://www.excalnor.com/directqb.htm>

Introduction



DirectQB is a game programming library written entirely in 386 assembly code for QuickBasic (QB) 4.5. It has been coded mainly to fill the void in the weak graphics, input and sound capabilities of QB. DirectQB uses screen mode 13h (VGA 320x200 with 256 colors - the common **SCREEN 13** for QB), supports keyboard, mouse and joysticks as input devices, and has a built-in sound engine that works with almost any Sound Blaster compatible sound card. It really has a lot of features, and this manual will help you make the most of them.

Installing DirectQB

The DirectQB library is distributed as freeware in a zipped (ZIP) distribution file. If you do not already have it, you can download it here:

<http://www.ecplusplus.com/files.php?file=http://www.ecplusplus.com/files/directqb.zip>

A number of other QuickBasic web sites have DirectQB, but it may not be the latest version.

Check the Hardware and System Requirements

Before you can use DirectQB, you must have certain hardware and software installed on your computer. The system requirements include:

- Any IBM®-compatible machine with an 80386 processor or higher.
- About 500 kilobytes free base (conventional) memory.
- A hard disk with a minimum of 1 megabyte available space for a full installation.
- A 5.25-inch or 3.5-inch disk drive, CD-ROM disc drive, or a connection to a remote computer to download DirectQB.
- A VGA-compatible video adapter capable of displaying 256 colors at 320x200 resolution.
- A fixed-frequency or multiple-frequency VGA compatible monitor.
- MS-DOS 5.0 or higher, or Windows 95/98/Me.
- Microsoft QuickBasic 4.5. DirectQB *will not* work with QBASIC.

Supported Hardware

DirectQB also supports the following additional hardware, but does not require it:

- Expanded memory (EMS), provided by HIMEM.SYS and EMM386.EXE, or another LIM 4.0 compatible EMS memory manager.
- A SoundBlaster 2.0 or compatible sound card.
- Microsoft compatible mouse and mouse driver.
- Any analog joystick or gamepad with 2 or 4 buttons, and most digital controllers.

Extracting the Archive

The DirectQB archive can be extracted using PKZip, WinZip, or any other program that supports standard ZIP files. DirectQB is drive and directory-independent, so you can install it on any drive and in any directory. The *-d* switch is not needed if you are using PKZip because the files can be in the same folder.

Setting Up EMM386.EXE

Although DirectQB does not require any EMS memory to function, EMS memory is needed in order to have off-screen buffers without running out of base memory fairly quickly. Each off-screen buffer is used as a “layer”, and each layer requires 64K of free base or EMS memory. In order to store layers in EMS memory, an EMS memory manager must be installed. MS-DOS and Windows ship with a file called EMM386.EXE, which can be used to provide EMS memory.

To configure your system to use EMM386.EXE, make sure these two lines are included at the beginning of your CONFIG.SYS file:

```
DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\EMM386.EXE amount RAM
```

Where *amount* is a number specifying the amount of EMS memory in KB to make available under DOS.

Note The example assumes you have Windows installed in the C:\WINDOWS directory. If you are using MS-DOS, substitute your DOS directory.

To set up your system with 4 MB of free EMS memory, you should add the following lines:

```
DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\EMM386.EXE 4096 RAM
```

After adding these lines, reboot your system, and EMS will be available and ready to be used by the library.

Setting Up the Sound Blaster Card

In order to use DirectQB's sound engine, you will need at least a SoundBlaster 2.0 compatible sound card. This is because DirectQB uses the "autoinit" mode for DMA transfers and requires a DSP chip version 2.00 or better. If you want to use **DQBinstallSB** with auto detection mode, the **BLASTER** environment variable must be set. Add the following line to your AUTOEXEC.BAT file:

```
SET BLASTER = Aaaa Ii Dd
```

Where *aaa* is the hexadecimal base address of your sound card (commonly 220), *i* is the IRQ number and *d* is the DMA channel to be used. A common setting would be like this:

```
SET BLASTER = A220 I7 D1
```

This sets the base address as 220h, IRQ 7 and DMA channel 1. DirectQB supports any base address, IRQ 2, 3, 5 and 7, and DMA channels 0 to 3. There are other settings for the **BLASTER** variable, such as H and T, but they are not required for DirectQB.

Base Memory Requirements

Besides the necessary 500K required to run QuickBasic and your program, additional base memory may be needed if you wish to use all of the features that DirectQB offers.

The sound engine requires 16K of extra base memory if you want a custom volume level for each voice. If all voices use the same volume setting, this is not required.

The blender map system is another way to keep your base memory busy. The amount of memory needed to use a single blender map depends on the map itself, and can vary from 256 bytes up to 64K per map. You can have up to 10 blender maps active at the same time.

It is expected that you will run into an out of memory error if you call the DirectQB functions that require extra base memory without calling the **SETMEM** function first. See the QB online help (in the IDE) and the examples in this manual for details on how to use **SETMEM**.

Features

DirectQB version 1.61 has the following features.

Graphics

- Supports 320x200 256-color video mode, with an almost unlimited number of layers (off-screen buffers) stored in EMS memory or base memory. The library automatically handles EMS for you.
- All the functions draw on the screen as well as on the off-screen buffers.
- Several drawing primitives, including pset, line, gline, ellipse, box, full box, and paint.
- Fast sprite handling functions, compatible with standard **GET** and **PUT**. Also has support for sprite flipping, scaling, rotation, zooming, translucency, and direct sprite drawing from one layer to another without requiring an external QB array.
- Sprite collision detection with selectable method (bounding box or pixel perfect), and support for collision layers.
- Color blending routines to handle up to 10 customizable blender maps, allowing the creation of any color combination.
- Clipping is supported for almost all the graphical functions.
- Loads and saves images of any size (up to 320x200) in BSV, BMP, and PCX format.
- Plays FLI animation files.
- Font routines with customizable fonts, non-fixed size font support and custom styles, such as bold, italic, underlined, or any combination, with support for color blending and textured text.
- Smooth palette handling with routines to fade the current palette into any new one, as well as into any specified color, and to rotate colors.
- Multidirectional scrolling for an entire layer or only a portion of it.
- Transparent screen copy for parallax scrolling effects, blended layer copy for huge translucency effect.
- Fast triangle drawing primitives allowing flat-shading, gouraud-shading, and affine texture mapping (also with support for bilinear filtering), all with support for color blending.

Input

- Custom keyboard interrupt handler that returns the state (pressed or released) of any key at any time.
- Several keyboard handling functions working under this IRQ handler, to read a key, to get the ASCII code from its scan code, and more.

-
- Fast and easy joystick and gamepad routines, with auto detection and auto calibration, support for one or two 2-button controllers, or one 4-button controller.
 - Mouse variables (coordinates and buttons status) are automatically updated when the mouse is moved, allowing your program to know the actual mouse state at any time without calling other routines.
 - Mouse routines to change the cursor shape, mouse range, speed, and position.

Sound

- IRQ driven sound engine for fast and easy sound playback via DMA transfers.
- Loads and plays sound effects directly from EMS.
- Supports 8-bit mono WAV files up to 23,000 Hz.
- Customizable number of channels, from 1 to 32, for up to 32 sound effects playing simultaneously.
- Real-time sound resampling.
- Customizable volume setting for each of the voices.
- Customizable master volume setting.

Miscellaneous

- Free EMS for other program use.
- Bit handling routines, including read/set/reset/toggle bit, shift left and right.
- Routines to find the color in any specified palette that best fits with the specified red, green, and blue hues.
- Useful routine to find the angle between any two given points.
- Directory scanning routines, with support for Windows long filenames.
- Fast multi-purpose array record sorting routine.
- Internal high precision timer for frame synchronization.
- Error messaging system.
- Datafile decoding routines, with support for password-encrypted data.

Included Files

The DirectQB library is now shipped as a set of OBJ files, together with the assembly source code. You should have these files in the library directory, along with the corresponding assembly file:

File	Description
3D.OBJ	Triangle drawing functions
BIT.OBJ	Bit handling routines
BLENDING.OBJ	Color blending support module
DATAFILE.OBJ	Datafile encoding/decoding routines
DISK.OBJ	Disk and directory handling functions
DRAW.OBJ	Primitive drawing functions
FONT.OBJ	Font functions
IMAGE.OBJ	BSV/BMP/PCX image file handling
JOYSTICK.OBJ	Joystick/gamepad support module
KEYBOARD.OBJ	Custom keyboard handler routines
MAIN.OBJ	Holds main library functions and common variables
MOUSE.OBJ	Mouse handling routines
PALETTE.OBJ	Palette handling routines
SOUND.OBJ	Sound engine module
SPRITE.OBJ	Special sprite drawing routines

Together, these files make up the DirectQB library. We will examine how to work with them later. The following is a list of additional files you should have in your DirectQB directory (other than the ASM source files).

File	Description
DIRECTQB.DOC	Text-only library manual
DQBENC.BAS	Datafile encoder program
DQBMAN.BAS	DirectQB Library Manager utility
DQBMAN.MHL	Help file for the library manager
DQBTOOLS.BAS	Font and mouse cursor editor
ERRORS.INC	Include file used by MAIN.ASM; contains error messages
FILELIST.TXT	Complete list of all the files in the DirectQB package
HITECH.FNT	Another font created with DQB Tools
README.1ST	Welcome message
SCRIPT.FNT	Script font created with DQB Tools
SMALL.FNT	Small font created with DQB Tools

Accessing the Library

DirectQB is made up of several OBJ files, and this means you can link some of them to create your own quick library, depending on your needs. For example, if you are not planning to use the sound engine, there's no need to include the SOUND.OBJ file

in the library. The library manager utility allows you to select each of these DirectQB modules, and creates the following files depending on your selections.

File	Description
CALLS.BAS	Used during the library building process, can be deleted
DIRECTQB.BI	Include file for the DirectQB library
DQB.LIB	Quick library file to be linked to your executable programs
DQB.QLB	Quick library file to be used from inside the QB IDE
INSTALL.DAT	Data on installed modules for library manager
INSTALL.LOG	Installation log file

Note For a list of the functions contained within each of the OBJ modules, see Appendix F.

In order to use DirectQB with QuickBasic, you need to create the LIB and QLB files. The DirectQB Library Manager utility (DQBMAN.BAS) will guide you through this process. Run that program now!

Once the quick library is built, load QB with it at the DOS prompt by typing:

QB /L DQB

The IDE will appear, and you'll be able to use the DirectQB functions.

Note It is recommended that you also load QuickBasic with the **/AH** switch. This allows you to have arrays larger than 64K in your program, and will help eliminate "out of memory" errors.

Basic Concepts

The DirectQB library makes an extensive use of EMS memory to store off-screen buffers ("layers") and sounds. A layer is basically a screen buffer that is hidden from view. You can draw pixels on it, as well as use any of the graphical functions of this library. The EMS memory is automatically handled for you, so all you have to do when calling any graphical function is specify which layer to use. To draw something directly to the screen, just use layer number 0 (there is also a constant named VIDEO, as specified in Appendix A). You can specify the number of layers to allocate in EMS by calling the **DQBinit** function, as well as the number of sounds to allocate space for (again in EMS) and the amount of free EMS to reserve for your own purposes.

DQBinit must *always* be called before calling any other function of this library, or you'll probably crash your system. Remember to also call **DQBclose** just before ending your program; this will free previously allocated memory and turn off the custom keyboard handler and the sound engine if they were on (plus doing other

tasks). You can also avoid using EMS, by allocating up to 10 extra layers stored in base memory. To set up these "base layers", refer to the **DQBsetBaseLayer** function description in Chapter 2.

Other than that, you should also keep in mind the following things:

- To speed things up, a lot of checks are skipped. For example, if you initialize the library with 3 extra layers, *do not* refer to layer -1 or 4!
- When referring to base layers, be sure you've previously set up them, because if you draw something on a layer that has not be configured with the **DQBsetBaseLayer** function, you're probably going to crash your system.
- Every graphical function that draws something, except **DQBbox**, **DQBboxf**, **DQBfPut**, and **DQBfilterBox**, supports clipping. This means that every pixel that lies outside the current clipping box will not be drawn. The clipping box is set at startup (when calling **DQBinit**) to (0, 0)-(319, 199), and it can be changed at any time by calling the **DQBsetClipBox** function.
- The sprite data format for library "get" and "put" routines is the same format used by the QB standard **GET** and **PUT** statements. The advantage of using the DirectQB functions is that they work with layers, support clipping (except **DQBfPut**), and are faster than **GET** and **PUT**.
- **DQBxPut** has the great advantage that it does not require you to store the sprites in an array before they are drawn. This means you save extra memory for your application, but the downside is that **DQBxPut** is slower than normal **DQBput**, and it can only handle sprites with a height up to 50 pixels.
- The transparent color is color number 0 and cannot be changed.
- When calling a function that requires passing the upper-left (x1, y1) and lower-right (x2, y2) coordinates, always remember that it x1 must be less than x2 ($x1 < x2$) and y1 must be less than y2 ($y1 < y2$), or you'll probably crash your system.
- By default, when calling **DQBprint**, the standard BIOS font is used. This can be changed by calling the **DQBsetFont** procedure. To restore the BIOS font, just call **DQBsetBIOSfont**.
- When referring to a color hue, the red, green, and blue components must always be in the range 0-63, otherwise your system may crash.
- The blending functions all require a blender map. If a blender map has not been created by **DQBcreateBMap**, any calls to **DQBsetBMap**, **DQBgetBMap**, **DQBloadBMap**, **DQBsaveBMap**, **DQBbPut**, or **DQBfilterBox** will do nothing. You can create up to 10 blender maps. To create them, you'll need some free base memory. See **DQBcreateBMap** for details.
- Blender maps can map up to 255 foreground color entries. I suggest you leave color 0 unmapped, as it should always be used for transparency only.
- Creating a blender map is not an easy task. It depends on the current palette and what the effect you are trying to achieve. Once created, it is suggested that you

store your blender map in a file so that you can load it at any time, without having to repeat the slow process.

- The default collision detection method is the bounding box check.
- You cannot use a frame rate lower than 18; lower values will be rounded to this number. Also, when calling the **DQBsetFrameRate** function, the system clock rate is altered, and some programs (like TSRs - SBMIDI for example) that use the system timer may not run properly.
- When the custom keyboard handler has been turned on by calling the **DQBinstallKeyboard** function, the standard QB functions that handle the keyboard should not be used, or you'll probably lock up your machine. These functions can be called again only after restoring the old keyboard handler by calling **DQBremoveKeyboard**.
- If you want to use the mouse, after calling **DQBinit** and entering the VGA mode, it is recommended that you call the **DQBresetMouse** function. This will set the default cursor shape and range.
- The mouse status (coordinates and button status) is automatically updated when a change occurs; you don't have to call any other function to update it.
- The joystick is different: before calling the **DQBjoyMove** and **DQBjoyFire** functions, you have to update the internal joystick variables by calling the **DQBpollJoy** function. This is a relatively slow operation, especially if no joysticks are connected, so be warned.
- The sound engine still has limitations: it supports only 8-bit mono WAV files for now, and you need a sound card with a DSP 2.00 or better.
- Datafiles can store up to 256 objects. Each object is referred to by a unique ID number, and you can't use the same number for different objects.
- If you have problems decoding an object from a datafile, check your password first, then if it's correct, check to make sure you referred to the right object (i.e. check if the image you're trying to decode isn't stored as a sound).

One Last Thing...

When writing your program, remember to include this line at the very beginning of your code:

```
'$INCLUDE: 'DIRECTQB.BI'
```

This will include the file DIRECTQB.BI in your project. This file contains the function and constant declarations needed to use DirectQB.

Library Limitations

DirectQB operates only in 320x200 256-color video mode. I really don't know if I'll ever release a high resolution or a high color version. I have the knowledge to do it, but this would require me to rewrite almost the entire library.

The sound engine supports only 8-bit mono WAV files for now, and the sampling rate is limited to 23,000 Hz. Why, you ask? Well, adding support for higher frequencies requires a lot of extra code, and I don't think a game requires such high-quality sounds. In addition, 16-bit sounds would require twice the memory to store them, so why bother?

The routines are written with speed in mind, so a lot of range checks are skipped. Don't get angry if your computer crashes, because if it does, it's probably due to you.

DirectQB can handle up to two game controllers with 2 buttons each, or only one controller with 4 buttons. The routines support either gamepads (8-direction movement) or analog joysticks (variable x and y-axis positions). Digital controllers are supported if they have a driver that can emulate an analog signal.

Fonts can have a maximum size of 8x8 pixels. The height must be the same for all the characters, while the width may vary.

Datafiles are a valid alternative to having multiple small files in your game directory. The compression is not the best (a simple RLE algorithm is used), but it's really fast. The password can only be 8 characters maximum. To avoid encrypting your datafiles, specify a null password.

For the various file formats used by DirectQB, see Appendix C.

Function Reference



Note All the example programs in this section have been tested with the full library release, i.e. all the modules have been included. For efficiency reasons, you can remove unused modules at any time with the DirectQB Library Manager program.

DQBangle Function

Prototype `DECLARE FUNCTION DQBangle(BYVAL x1, BYVAL y1, BYVAL x2, BYVAL y2)`

Calling	Parameter	Description
	x1	x coordinate of first point
	y1	y coordinate of first point
	x2	x coordinate of second point
	y2	y coordinate of second point

Returns An integer value holding the angle between the two points, in the range 0-255.

Description Have you ever wanted to know the angle an enemy should face to look for the player in your games? This function lets you know the exact angle between two given points, i.e. what angle the object located at the first point should face to "see" the second one. Returned angle is in the range 0-255, where 0 means the second point is above the first, 127 that it's below it, etc.

Notes As **DQBangle** returns the angle in the range 0-255, you can easily use this function in conjunction with **DQBrPut**.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Allocate memory for a simple 32x32 pixel sprite
DIM Sprite(514)

' Initialize the library with one layer, no sounds, and no user EMS
```

```

IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Check if a mouse is available
IF NOT DQBmouseDetected THEN      ' Mouse not found!
    PRINT "This example requires a mouse to run!"
    DQBclose
    END
END IF

DQBinitVGA
DQBclearLayer 1

' Draw our sprite and get it from a hidden layer
DQBline 1, 15, 0, 22, 26, 40
DQBline 1, 15, 0, 8, 26, 40
DQBline 1, 8, 26, 22, 26, 40
DQBpaint 1, 16, 16, 4
DQBget 1, 0, 0, 31, 31, VARSEG(Sprite(0)), VARPTR(Sprite(0))

' Start demo
DQBresetMouse
DQBmouseShow
DO
    ' Clear layer 1 for double buffering effect
    DQBclearLayer 1

    ' Find angle between the center of our sprite and the mouse cursor
    a = DQBangle(160, 100, DQBmouseX, DQBmouseY)

    ' Draw rotated sprite
    DQBrPut 1, 144, 84, VARSEG(Sprite(0)), VARPTR(Sprite(0)), a, 100

    ' Update the screen
    DQBwait 1
    DQBmouseHide
    DQBcopyLayer 1, VIDEO
    DQBmouseShow

    ' End demo if the user presses a key or any of the mouse buttons
    IF DQBmouseLB OR DQBmouseRB OR INKEY$ <> "" THEN EXIT DO
LOOP

' End the program
DQBclose
END

```

DQBasc Function

Prototype DECLARE FUNCTION DQBasc(BYVAL ScanCode, BYVAL ShiftFlag)

Calling	Parameter	Description
---------	-----------	-------------

ScanCode	Scan code of the character
ShiftFlag	See function description

Returns	An integer value holding the ASCII code number of specified character.
Description	This function is used to retrieve the ASCII code of a specified character, when you know its scan code. The problem is that for each scan code there can be one or two associated characters. For example, the "A" character is associated with the scan code number 30, but "a" is also associated with 30. DirectQB has two internal tables holding all the possible combinations, so you must specify in which table to search for specified character. That's the purpose of the ShiftFlag parameter: if true, in our example you'll get an "A", otherwise an "a".
Notes	This function can be called even if the keyboard handler is off.
Example	None.

DQBbgtri Sub

Prototype DECLARE SUB DQBbgtri (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL c1, _
BYVAL x2, BYVAL y2, BYVAL c2, BYVAL x3, BYVAL y3, BYVAL c3)

Calling	Parameter	Description
	Layer	Layer to draw the triangle on
	x1	x coordinate of the first vertex
	y1	y coordinate of the first vertex
	c1	color of the first vertex
	x2	x coordinate of the second vertex
	y2	y coordinate of the second vertex
	c2	color of the second vertex
	x3	x coordinate of the third vertex
	y3	y coordinate of the third vertex
	c3	color of the third vertex

Returns None.

Description Draws a gouraud-shaded triangle with (x1, y1), (x2, y2), and (x3, y3) as vertices, interpolating specified colors, using the current palette. Resulting colors are finally blended with the background, using the current blender map. **DQBbgtri** does not support transparency, but it's affected by the clipping box.

Notes Gouraud shading works better when you set a palette with lots of shades of the same color; it is up to you to find the best settings for your needs. Remember to also set an appropriate blender map to obtain the best results. See also **DQBtri**, **DQBbtri**, and **DQBttri**

Example None.

DQBbox Sub

Prototype DECLARE SUB DQBbox (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, _
BYVAL y2, BYVAL Col)

Calling	Parameter	Description
	Layer	Layer to draw the box on
	x1	Upper left corner x coordinate
	y1	Upper left corner y coordinate
	x2	Lower right corner x coordinate
	y2	Lower right corner y coordinate
	Col	Drawing color

Returns None.

Description Draws an empty box on the given layer, with (x1, y1) and (x2, y2) as the upper left and lower right corners, with Col color.

Notes This function is not affected by the clipping box, and no range checks are done, so pay attention. In addition, remember that $x1 < x2$ and $y1 < y2$.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Draw a box on the screen
DQBbox VIDEO, 0, 0, 319, 199, 15

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END
```

DQBboxf Sub

Prototype DECLARE SUB DQBboxf (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, _
BYVAL y2, BYVAL Col)

Calling	Parameter	Description
	Layer	Layer to draw the box on
	x1	Upper left corner x coordinate
	y1	Upper left corner y coordinate
	x2	Lower right corner x coordinate
	y2	Lower right corner y coordinate
	Col	Drawing color

Returns None.

Description Same as **DQBbox**, but draws a full box filled with Col color.

Notes See **DQBbox**.

Example See **DQBbox** example.

DQBbPut Sub

Prototype DECLARE SUB DQBbPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff, BYVAL BMap)

Calling	Parameter	Description
	Layer	Layer to draw the sprite on
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)
	BMap	Blender map to be used

Returns None.

Description **DQBbPut** draws a sprite, blending its colors with the background. The way the colors are blended depends on the specified blender map. If the blender map has not been created by calling **DQBcreateBMap**, this function does nothing. There are virtually an unlimited number of special color effects allowed by using the blender map system. For example, by setting up an appropriate blender map for your palette, you can obtain a translucency effect.

Notes

DQBbPut is affected by the clipping box, so pixels outside this box will not be drawn; transparency is also supported. See also **DQBcreateBMap**, **DQBsetBMap**, **DQBgetBMap**, **DQBloadBMap**, **DQBsaveBMap**, **DQBput**, **DQBfPut**, **DQBspPut**, and **DQBBrPut**.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with one layer, no sounds, and no user EMS
IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Deallocate 256 bytes from the far heap to allow DQBcreateBMap to
' allocate base memory for the blender map. We're going to map one
' color only, so we need 1 * 256 = 256 bytes to create the blender
' map.
AllocMem& = SETMEM(-256)

' Build blender map 1, mapping color 40 only
IF DQBcreateBMap(1, 40, 40) THEN DQBclose: PRINT DQBerror$: END

' Here we set our only color combination on bmap number 1: If a pixel
' of color 40 (bright red) is drawn over another of color 32 (bright
' blue), the pixel is drawn with color 13 (bright purple)
DQBsetBMap 1, 40, 32, 13

' Dimension our two sprites using DQBsize
size = DQBsize(0, 0, 63, 7)
DIM Sprite(size)

DQBclearLayer 1
DQBprint 1, "DirectQB", 0, 0, 32
DQBprint 1, "Power!", 0, 8, 40

' Get our sprites
DQBget 1, 0, 0, 63, 7, VARSEG(Sprite(0)), VARPTR(Sprite(0))
DQBget 1, 0, 8, 63, 15, VARSEG(Sprite(0)), (VARPTR(Sprite(0)) + size)

DQBinitVGA

' Begin the demo
FOR x = -64 TO 320
  ' Clear our hidden layer
  DQBclearLayer 1

  ' Draw our sprites on layer 1
  DQBput 1, x, 96, VARSEG(Sprite(0)), VARPTR(Sprite(0))
  DQBbPut 1, (256 - x), 96, VARSEG(Sprite(0)), (VARPTR(Sprite(0)) _
+ size), 1

  ' Copy layer 1 on the screen
  DQBwait 3
  DQBcopyLayer 1, VIDEO
NEXT x

' End the program
DQBclose
FreeMem& = SETMEM(256)
```

 END

DQBbtri Sub

Prototype DECLARE SUB DQBbtri (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, _
BYVAL y2, BYVAL x3, BYVAL y3, BYVAL Col, BYVAL BMap)

Calling	Parameter	Description
	Layer	Layer to draw the triangle on
	x1	x coordinate of the first vertex
	y1	y coordinate of the first vertex
	x2	x coordinate of the second vertex
	y2	y coordinate of the second vertex
	x3	x coordinate of the third vertex
	y3	y coordinate of the third vertex
	Col	Color
	BMap	Blender map to be used

Returns None.

Description Draws a triangle with (x1, y1), (x2, y2), and (x3, y3) as vertices, and fills it by blending the specified color with the background, using the specified blender map. This function does not support transparency, but it's affected by the clipping box.

Notes See also **DQBtri**, **DQBgtri**, and **DQBttri**.

Example None.

DQBbttri Sub

Prototype DECLARE SUB DQBbttri (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, _
BYVAL y2, BYVAL x3, BYVAL y3, BYVAL u1, BYVAL v1, BYVAL u2, BYVAL _
v2, BYVAL u3, BYVAL v3, BYVAL TextureSeg, BYVAL TextureOff, BYVAL _
BMap)

Calling	Parameter	Description
	Layer	Layer to draw the triangle on
	x1	x coordinate of the first vertex
	y1	y coordinate of the first vertex
	x2	x coordinate of the second vertex
	y2	y coordinate of the second vertex
	x3	x coordinate of the third vertex
	y3	y coordinate of the third vertex
	u1	x coordinate of first vertex on texture
	v1	y coordinate of first vertex on texture
	u1	x coordinate of second vertex on texture
	v1	y coordinate of second vertex on texture
	u1	x coordinate of third vertex on texture
	v1	y coordinate of third vertex on texture
	TextureSeg	Array segment holding the texture (use VARSEG)
	TextureOff	Array offset holding the texture (use VARPTR)
	BMap	Blender map to be used
Returns	None.	
Description	Draws a texture-mapped triangle with (x1, y1), (x2, y2), and (x3, y3) as vertices. This function works exactly like DQBttri , but the colors are blended with the background by using the specified blender map.	
Notes	See also DQBttri , DQBsetTextureSize , DQBtri , DQBbtri , and DQBgtri .	
Example	None.	

DQBchDir Function

Prototype	DECLARE FUNCTION DQBchDir (NewDir AS STRING)	
Calling	Parameter	Description
	NewDir	New directory
Returns	0 if the operation is successful, otherwise -1.	
Description	This function changes the current directory to the new specified one. You can also use Windows long directory names.	
Notes	The main difference from the standard QB statement CHDIR is that DQBchDir also works with Windows long filenames, while CHDIR doesn't.	
Example	None.	

DQBclearLayer Sub

Prototype DECLARE SUB DQBclearLayer (BYVAL Layer)

Calling **Parameter** **Description**

Layer Layer to clear

Returns None.

Description Clears the contents of the given layer; all the pixels on the layer are set to color 0. This function is automatically called for all the allocated layers when calling **DQBinit**.

Notes None.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

PRINT "Press a key to clear the screen!"
WHILE INKEY$="":WEND

' Clear the screen
DQBclearLayer VIDEO

' End the program
DQBclose
END
```

DQBclose Sub

Prototype DECLARE SUB DQBclose ()

Calling None.

Returns None.

Description This function deallocates the EMS memory used by the layers initialized by **DQBinit** and turns off the keyboard interrupt handler and the sound engine if they were on. It also deallocates the blender maps if they were created, and does other closing stuff you may have forgotten. **DQBclose** must always be called before ending your

program, otherwise allocated EMS memory will be lost, and you'll have to reboot your computer to free it.

Notes If your program is in graphics mode, this function also automatically switches the screen to plain text mode, so you don't have to worry about calling **DQBinitText** before it. See also **DQBinit**.

Example None.

DQBcloseDataFile Sub

Prototype DECLARE SUB DQBcloseDataFile ()

Calling None.

Returns None.

Description Closes the open datafile. If no datafile is open, this function does nothing.

Notes You cannot have more than one open datafile at the same time, so this function may be useful when you have your data stored in different datafiles. This function is automatically called by **DQBclose**. See also **DQBopenDataFile**.

Example See **DQBopenDataFile** example.

DQBcloseFLI Sub

Prototype DECLARE SUB DQBcloseFLI ()

Calling None.

Returns None.

Description Closes a FLI file that was previously opened with the **DQBopenFLI** function. You must close a FLI file with this function before attempting to open another one, as DirectQB can handle only one opened FLI animation at a time. If a FLI file was not open, this function does nothing.

Notes This function is meant to be used only in conjunction with **DQBopenFLI** and **DQBplayFLIstep**. If you're going to play a whole FLI animation "as is" by calling **DQBplayFLI**, there's no need to call this function when the animation ends. **DQBclose** automatically calls this function.

Example See **DQBplayFLIstep** example.

DQBcollide Function

Prototype DECLARE SUB DQBcollide (BYVAL x1, BYVAL y1, BYVAL Seg1, BYVAL _
Off1, BYVAL x2, BYVAL y2, BYVAL Seg2, BYVAL Off2)

Calling	Parameter	Description
	x1	x position of first sprite
	y1	y position of first sprite
	Seg1	Segment of first sprite array (use VARSEG)
	Off1	Offset of first sprite array (use VARPTR)
	x2	x position of second sprite
	y2	y position of second sprite
	Seg2	Segment of second sprite array (use VARSEG)
	Off2	Offset of second sprite array (use VARPTR)

Returns True if the sprites collide, otherwise false.

Description Given two sprites, this function tests if they collide, using the current check method. There are two collision detection methods available: bounding box check and pixel-perfect check. Needless to say, the first is the fastest, but the other gives you a perfect collision precision.

Notes By default, the collision detection method is the bounding box check. You can change this at any time by calling **DQBsetCollideMethod**.

Example None.

DQBcollideOnLayer Function

Prototype DECLARE SUB DQBcollideOnLayer (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff)

Calling	Parameter	Description
	Layer	Layer where to test for collision(s)
	x	x position of sprite
	y	y position of sprite
	SpriteSeg	Segment of sprite array (use VARSEG)
	SpriteOff	Offset of sprite array (use VARPTR)

Returns Pixel color of first colliding object, or 0 if no collisions occur.

Description This function introduces the concept of a *collision layer*. By using a collision layer, you can easily achieve fast collision detections between several sprites. After

specifying a layer, and the sprite position and data, **DQBcollideOnLayer** returns the color of the first non-transparent pixel on the layer that actually collides with the specified sprite. If no collision was detected, 0 is returned.

Notes Sprite position can be inside as well as outside layer space (i.e. clipping is supported). It is suggested that you use **DQBhPut** to draw all your active sprites on the collision layer, so you can easily check for collisions using this function.

Example None.

DQBcopyBlendLayer Sub

Prototype DECLARE SUB DQBcopyBlendLayer (BYVAL SourceLayer, BYVAL DestLayer, _ BYVAL BMap)

Calling	Parameter	Description
	SourceLayer	Source layer to copy data from
	DestLayer	Destination layer to copy data onto
	BMap	Blender map to be used

Returns None.

Description Copies the contents of SourceLayer and blends them onto DestLayer, using the specified blender map.

Notes See also **DQBcopyLayer**, as this function works almost the same way.

Example None.

DQBcopyHitLayer Sub

Prototype DECLARE SUB DQBcopyHitLayer (BYVAL SourceLayer, BYVAL DestLayer, _ BYVAL Col)

Calling	Parameter	Description
	SourceLayer	Source layer to copy data from
	DestLayer	Destination layer where to copy data
	Col	Fill color

Returns None.

Description	Works exactly like DQBcopyLayer , except all the non-0 pixels of the source layer are drawn onto the destination layer with the same color specified with the "Col" parameter.
Notes	This is useful to create collision layers, to be used in conjunction with the DQBcollideOnLayer function.
Example	None.

DQBcopyLayer Sub

Prototype DECLARE SUB DQBcopyLayer (BYVAL SourceLayer, BYVAL DestLayer)

Calling	Parameter	Description
	SourceLayer	Source layer to copy from
	DestLayer	Destination layer to copy to

Returns None.

Description Copies the contents of SourceLayer onto DestLayer. All the data previously on DestLayer will be lost. This function can be used to create double-buffered animations. Just draw every sprite on a layer in EMS, and then use **DQBcopyLayer** to copy its contents to the screen.

Notes None.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'

' Initialize the library with one layer, no sounds, and no user EMS
IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Draw a box on the hidden layer
DQBbox 1, 0, 0, 319, 199, 15

' Copy the hidden layer on the screen
DQBcopyLayer 1, VIDEO

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END
```

DQBcopyTransLayer Sub

Prototype `DECLARE SUB DQBcopyTransLayer (BYVAL SourceLayer, BYVAL DestLayer)`

Calling	Parameter	Description
	SourceLayer	Source layer to copy data from
	DestLayer	Destination layer to copy data onto

Returns None.

Description This function is similar to **DQBcopyLayer**, but the pixels with color 0 of the source layer are skipped and are not copied onto the destination layer. This allows for transparent layer copying. With some practice, it is possible to obtain a parallax scrolling effect using this function with **DQBscroll**.

Notes Needless to say, **DQBcopyTransLayer** is slower than **DQBcopyLayer**.

Example See **DQBcopyLayer** example.

DQBcreateBMap Function

Prototype `DECLARE FUNCTION DQBcreateBMap (BYVAL BMap, BYVAL FirstCol, BYVAL _ LastCol)`

Calling	Parameter	Description
	BMap	Code number of the blender map to create
	FirstCol	First foreground color to map
	LastCol	Last foreground color to map

Returns	Integer	Description
	0	Blender map successfully created and initialized
	1	Not enough free base memory
	2	Blender map already created

Description This function attempts to create a blender map in *base* memory. This requires as much memory as the number of foreground colors you map. A blender map is a table with all the possible color combinations between the specified foreground colors and the 256 background colors. This is the formula to obtain the amount of memory needed to create a blender map:

$$(LastCol - FirstCol + 1) * 256$$

The foreground color is the color of each pixel of your sprite, and the background color is the color of each pixel of the background layer where you're drawing.

Suppose you have a palette with color 1 as bright red, color 2 as bright green, and color 3 as bright purple. You could call:

```
DQBsetBMap 7, 1, 2, 3
```

This way, every time you use **DQBbPut** referring to blender map number 7 to draw a sprite with some pixels of color 1, and the background is filled with color 2, you'll see a cyan sprite (drawn with color 3).

Blender maps are a difficult concept, but once you understand them, you'll also understand that the system offers virtually any special effect, depending on your palette and, of course, on the blender map you set.

Notes

You have to set your blender map based on your own needs. As it's often a slow process, you can use the **DQBsaveBMap** and **DQBloadBMap** functions to help you save time. It is also suggested that you use **DQBfindCol** to create your blender map independently from the current palette. Also keep in mind that any blending function will not work if the blender map has not been created by calling **DQBcreateBMap**, and remember that to free more free base memory you should always call the QB function **SETMEM**. Another important thing to remember is that a blender map can only have up to 255 foreground colors mapped. Do not try to create a map with 256 elements, or you may end up with unpredictable results. I suggest you leave color 0 unmapped, since it should always be used for transparency only. See also **DQBsetBMap**, **DQBgetBMap**, **DQBloadBMap**, **DQBsaveBMap**, **DQBbPut**, and **DQBfilterBox**.

Example

See **DQBbPut** example.

DQBdir\$ Function

Prototype `DECLARE FUNCTION DQBdir$ (Mask AS STRING, Attrib AS INTEGER)`

Calling

Parameter	Description
Mask	Mask for file(s) to search for
Attrib	Attributes of the file(s) to search for

Returns

A string holding the filename of the next file found.

Description

DQBdir\$ scans the directory for the files matching specified mask. The mask can include a full path and wildcards, and you can also use Windows long filenames. If no files are found, the function returns a null string.

The first time **DQBdir\$** is called with a mask, it searches for the first file matching it. If a file is found, then you can continue to scan the directory by calling this function again, but by passing a null string as mask. Repeat this process until you get a null string, and you'll obtain a full directory scan.

The **attrib** parameter specifies which kind of files to search for. There are constants defined for you in **DIRECTQB.BI**, which you can also add together to search for a larger variety of files (for example, to search simultaneously for normal archive files and directories); see Appendix A for the list.

Notes

Here's a hint: you can determine if a file **Filename\$** exists by simply checking if **DQBdir\$(ATTRIB.A, Filename\$)** gives a null string or the filename itself. If you get a null string, the file does not exist.

See also **DQBdrive\$, DQBpath\$, DQBnumDrives, DQBsetDrive, and DQBchDir.**

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

CLS

' Print available drives
PRINT "Available drives: ";
FOR i = 1 TO DQBnumDrives
    PRINT "[" + CHR$(64 + i) + "] ";
NEXT i
PRINT

' Print volume label
PRINT "Current volume label is " + DQBdir$("*.\"", ATTRIB.L)

' Print current drive and directory
PRINT "Directory of " + DQBdrive$ + ":\\" + DQBpath$

' Search for normal archive ".BAS" files
File$ = DQBdir$("*.BAS", ATTRIB.A)
IF File$ = "" THEN
    ' No files matching
    PRINT "No .BAS files found!"
ELSE
    ' The first file was found! Let's print it.
    PRINT File$

    ' Now we search for the next files:
    DO
        ' The second time we don't need the mask or the attributes
        File$ = DQBdir$("", 0)

        ' If no more files are found, then exit the loop
        IF File$ = "" THEN EXIT DO
```

```

        ' Print our next file
        PRINT File$
    LOOP
END IF

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBdrive\$ Function

Prototype	DECLARE FUNCTION DQBdrive\$ ()
Calling	None.
Returns	The current drive letter character.
Description	This function simply returns a one character string holding the current drive letter.
Notes	See also DQBsetDrive , DQBnumDrives , DQBchDir , DQBdir\$, and DQBpath\$.
Example	See DQBdir\$ example.

DQBellipse Sub

Prototype	DECLARE SUB DQBellipse (BYVAL Layer, BYVAL x, BYVAL y, BYVAL rx, _ BYVAL ry, BYVAL Col)	
Calling	Parameter	Description
	Layer	Source layer to copy data from
	x	x coordinate of the center of your ellipse
	y	y coordinate of the center of your ellipse
	rx	Horizontal radius in pixels
	ry	Vertical radius in pixels
	Col	Ellipse color
Returns	None.	
Description	This function is draws ellipses (and, needless to say, a circle is an ellipse with equal radii). The radius can range from 0 to 256.	
Notes	DQBellipse is affected by the clipping box.	
Example	' All integers for speed	

```

DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'

' Initialize the library with one layer, no sounds, and no user EMS
IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

FOR i = 10 TO 90 STEP 10
  ' Draw our ellipses
  DQBellipse VIDEO, 160, 100, 90, i, 40
  DQBellipse VIDEO, 160, 100, i, 90, 40
NEXT i

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBerror\$ Function

Prototype DECLARE FUNCTION DQBerror\$ ()

Calling None.

Returns A string holding the last error message.

Description With this version of **DQBerror\$**, you do not need to check for every unsuccessful result of your DirectQB function calls anymore. When an error occurs, **DQBerror\$** automatically holds a short error message that explains what went wrong.

Notes None.

Example

```

' All integers for speed
DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'

' Initialize the library with one layer, one sound, and no user EMS
IF DQBinit(1, 1, 0) THEN DQBclose: PRINT DQBerror$: END

' If we get here, no errors have occurred
PRINT "DirectQB successfully initialized!"

' End the program
DQBclose
END

```

DQBfadeIn Sub

Prototype DECLARE SUB DQBfadeIn (Pal AS STRING)

Calling	Parameter	Description
	Pal	A string of 768 characters holding the palette to fade into
Returns	None.	
Description	Fades the current palette into the given one. The current palette may be completely different from the target one. The Pal string can be obtained by calling the DQBgetPal or the DQBloadImage functions, or it can be set manually. Keep in mind that each color has three bytes, representing the red, green, and blue components, in the range 0-63.	
Notes	See also DQBfadeTo , DQBgetPal , DQBsetPal , DQBpalOff , DQBgetCol , and DQBsetCol .	
Example	See DQBfadeTo example.	

DQBfadeStepIn Sub

Prototype DECLARE SUB DQBfadeStepIn (Pal AS STRING)

Calling	Parameter	Description
	Pal	A string of 768 characters holding the palette to fade into
Returns	None.	
Description	Works exactly like DQBfadeIn , but only fades all the colors by a single step. To make sure all your colors fade to the new palette, you should call this function 64 times repeatedly. DQBfadeStepIn can be useful to move objects around on the screen while fading.	
Notes	See also DQBfadeIn .	
Example	None.	

DQBfadeStepTo Sub

Prototype DECLARE SUB DQBfadeStepTo (BYVAL Red, BYVAL Green, BYVAL Blue)

Calling	Parameter	Description
	Red	Red hue to fade colors into
	Green	Green hue to fade colors into
	Blue	Blue hue to fade colors into

Returns None.

Description Works exactly like **DQBfadeTo**, but this function will only fade all the colors by a single step. As with **DQBfadeStepIn**, to make sure all your colors fade to the new palette, you should call this function 64 times. **DQBfadeStepTo** can be useful to move your stuff around while fading.

Notes See also **DQBfadeTo**.

Example None.

DQBfadeTo Sub

Prototype DECLARE SUB DQBfadeTo (BYVAL Red, BYVAL Green, BYVAL Blue)

Calling	Parameter	Description
	Red	Red hue to fade colors into
	Green	Green hue to fade colors into
	Blue	Blue hue to fade colors into

Returns None.

Description Fades all the colors in current palette to the specified red, green, and blue hues. To fade a palette to black, just call this function with 0, 0, 0 as parameters. This function is quite powerful, as it can fade any palette to any specified color.

Notes See also **DQBfadeIn**, **DQBgetPal**, **DQBsetPal**, **DQBpalOff**, **DQBgetCol**, and **DQBsetCol**.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

DIM Pal AS STRING * 768
```

```

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Store the default palette into Pal
DQBgetPal Pal

' Set all the colors to black
DQBpalOff

' Draw 500 pixels randomly on the screen
FOR i = 1 TO 500
    RANDOMIZE TIMER
    DQBpset VIDEO, (RND * 320), (RND * 200), (RND * 256)
NEXT i

' Fade the palette into the original one
DQBfadeIn Pal

' Fade the palette to white
DQBfadeTo 63, 63, 63

' and then to black
DQBfadeTo 0, 0, 0

' Restore the original palette
DQBsetPal Pal

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBfilterBox Sub

Prototype DECLARE SUB DQBfilterBox (BYVAL Layer, BYVAL x1, BYVAL y2, BYVAL _
x2, BYVAL y2, BYVAL Col, BYVAL BMap)

Calling	Parameter	Description
	Layer	Layer to draw the filter box on
	x1	Upper left corner x coordinate
	y1	Upper left corner y coordinate
	x2	Lower right corner x coordinate
	y2	Lower right corner y coordinate
	Col	Filter color
	BMap	Blender map to be used

Returns None.

Description	DQBfilterBox draws a full box, blending the specified color with the background, using the given blender map. If the blender map has not been created, this function will not draw the box.
Notes	The filter color acts as the foreground color; it's up to you to set an appropriate blender map. Remember $x2 \geq x1$ and $y2 \geq y1$. See also DQBcreateBMap , DQBloadBMap , DQBsaveBMap , DQBsetBMap , DQBgetBMap , and DQBbPut .
Example	None.

DQBfindCol Function

Prototype	DECLARE FUNCTION DQBfindCol (BYVAL Red, BYVAL Green, BYVAL Blue)	
Calling	Parameter	Description
	Red	Red hue of the color to find
	Green	Green hue of the color to find
	Blue	Blue hue of the color to find
Returns	An integer value ranging 0-255, holding the index of the color found.	
Description	DQBfindCol searches the current palette for the color that best fits with the specified red, green, and blue hues. It is useful when creating a blender map in conjunction with the DQBsetBMap routine.	
Notes	This function may return bad results, depending on the current palette and the specified hues, so use it with caution. As it scans the palette, you will see the screen flickering a little during each call to this function. To avoid this, you can use the DQBfindPalCol , which searches for a color in any specified palette.	
Example	None.	

DQBfindPalCol Function

Prototype	DECLARE FUNCTION DQBfindPalCol (Pal AS STRING, Red AS INTEGER, _ Green AS INTEGER, Blue AS INTEGER)	
Calling	Parameter	Description
	Pal	Palette to scan
	Red	Red hue of the color to find
	Green	Green hue of the color to find
	Blue	Blue hue of the color to find

Returns	An integer value ranging 0-255, holding the index of the color found.
Description	This function works exactly like DQBfindCol , except that it searches for a color in a specified palette, not necessarily on the current palette. This also avoids the bad looking flickering of the DQBfindCol function. The palette should be stored by DQBgetPal , DQBloadImage , or similar routines that use the standard DirectQB palette format (see Appendix C for details).
Notes	See also DQBfindCol .
Example	None.

DQBfPut Sub

Prototype DECLARE SUB DQBfPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff)

Calling	Parameter	Description
	Layer	Layer to draw the sprite on
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)

Returns None.

Description Like **DQBput**, **DQBfPut** draws a sprite on a specified layer, but using a faster algorithm. This function does not allow clipping or transparency, and works better with big sprites with a width that is a multiple of 4.

Notes See also **DQBput**, **DQBsPut**, **DQBrPut**, and **DQBbPut**.

Example See **DQBput** example.

DQBframeReady Function

Prototype DECLARE FUNCTION DQBframeReady ()

Calling None.

Returns True if enough time has passed to synchronize graphics, otherwise false.

Description	DQBframeReady can be used in conjunction with DQBsetFrameRate to synchronize your graphics with a specified frame rate. Once you set the frame rate with DQBsetFrameRate , you should wait until this function returns true before displaying each frame.
Notes	See DQBsetFrameRate .
Example	None.

DQBfttri Sub

Prototype DECLARE SUB DQBfttri (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, BYVAL y2, BYVAL x3, BYVAL y3, BYVAL u1, BYVAL v1, BYVAL u2, BYVAL v2, BYVAL u3, BYVAL v3, BYVAL TextureSeg, BYVAL TextureOff)

Calling	Parameter	Description
	Layer	Layer to draw the triangle on
	x1	x coordinate of the first vertex
	y1	y coordinate of the first vertex
	x2	x coordinate of the second vertex
	y2	y coordinate of the second vertex
	x3	x coordinate of the third vertex
	y3	y coordinate of the third vertex
	u1	x coordinate of first vertex on texture
	v1	y coordinate of first vertex on texture
	u2	x coordinate of second vertex on texture
	v2	y coordinate of second vertex on texture
	u3	x coordinate of third vertex on texture
	v3	y coordinate of third vertex on texture
	TextureSeg	Array segment holding the texture (use VARSEG)
	TextureOff	Array offset holding the texture (use VARPTR)

Returns None.

Description Works exactly like **DQBttri**, but applies bilinear filtering during the texturing process. This function is slower than **DQBttri**, but it draws better-looking triangles. You will need a gradient palette in order to achieve good results.

Notes As this function has been coded thinking about speed, it isn't too accurate with its calculations. To be sure your triangle will be drawn correctly, you should use a gradient palette set in the way that at least 4 or 5 colors are *not* used by your texture before and after its colors range. For example, if your texture uses only color 5 to 100, you also need to set colors 0-4 and 101-105 to avoid problems. See also **DQBsetTextureSize**, **DQBttri**, **DQBbtri**, **DQBgtri**, and **DQBttri**.

Example

```

' All integers for speed
DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'
DIM texture(2049)

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Create a red gradient palette
FOR i = 0 TO 63
    DQBsetCol i, i, 0, 0
    DQBsetCol 64 + i, 63, i, i
NEXT i

' Create a texture filled with random red pixels
FOR x = 0 TO 63
    FOR y = 0 TO 63
        DQBpset VIDEO, x, y, 5 + (RND * 118)
    NEXT y
NEXT x
DQBbox VIDEO, 0, 0, 63, 63, 127
DQBline VIDEO, 63, 0, 0, 63, 127

' Get the texture
DQBget VIDEO, 0, 0, 63, 63, VARSEG(texture(0)), VARPTR(texture(0))

' Clear the screen
DQBclearLayer VIDEO

' Draw 500 random triangles
FOR i = 1 TO 500
    RANDOMIZE TIMER
    DQBfttri VIDEO, RND * 320, RND * 200, RND * 320, RND * 200, RND *
    320, RND * 200, 0, 0, 63, 0, 0, 63, VARSEG(texture(0)), _
    VARPTR(texture(0))
NEXT i

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBget Sub

Prototype DECLARE SUB DQBget (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, _
 BYVAL y2, BYVAL SpriteSeg, BYVAL SpriteOff)

Calling	Parameter	Description
	Layer	Layer to get the sprite from
	x1	Upper left corner x coordinate
	y1	Upper left corner y coordinate
	x2	Lower right corner x coordinate
	y2	Lower right corner y coordinate
	SpriteSeg	Segment of the array where to store the sprite (use VARSEG)
	SpriteOff	Offset of the array where to store the sprite (use VARPTR)
Returns	None.	
Description	Gets a sprite from the specified layer in the box (x1, y1)-(x2, y2) and places the data into the specified array. The following formula can be used with integer arrays to determine the number of dimensions needed to store the image: $(((\text{Width} * \text{Height}) \setminus 2) + 4)$ You can also use the DQBsize function and divide the result by 2 to retrieve the size of your array without worrying about this formula. Sprites stored with DQBget can be then used by DQBput to create animations.	
Notes	DQBget is not affected by the clipping box. Also, $x1 < x2$ and $y1 < y2$. This routine is optimized for sprite widths that are a multiple of 2 or 4. Several sprites can be stored in the same array; just pass the array with a different index. In addition, the format of DQBget and DQBput is compatible with QB's GET and PUT statements, so you can use them with the same sprite arrays. See also DQBfPut, DQBsPut, DQBrPut, and DQBbPut.	
Example	See DQBput example.	

DQBgetBMap Function

Prototype DECLARE FUNCTION DQBgetBMap (BYVAL BMap, BYVAL ForeCol, BYVAL _ BackCol)

Calling	Parameter	Description
	BMap	Blender map to get color info from
	ForeCol	Foreground color
	BackCol	Background color
Returns	An integer value holding the color set for the specified combination in the specified blender map.	

Description Returns the color set in the specified blender map for the color combination given by the specified foreground and background colors. If the foreground color is not mapped into the blender map, this function may return unpredictable results.

Notes See also **DQBcreateBMap** and **DQBsetBMap**.

Example None.

DQBgetCol Sub

Prototype DECLARE SUB DQBgetCol (BYVAL Col, Red, Green, Blue)

Calling	Parameter	Description
	Col	Color index to retrieve data from
	Red	Integer variable to store the red hue in
	Green	Integer variable to store the green hue in
	Blue	Integer variable to store the blue hue in

Returns None.

Description Gets the red, green, and blue hues of the specified color and stores them in the Red, Green and Blue integer variables. Hues range from 0-63.

Notes See also **DQBsetCol**.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'
DIM r, g, b

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Get the hues for color 2 (dark green in default BIOS palette)
DQBgetCol 2, r, g, b

PRINT "Color 2 hues:"
PRINT "Red:", r
PRINT "Green:", g
PRINT "Blue:", b

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END
```

DQBgetPal Sub

Prototype DECLARE SUB DQBgetPal (Pal AS STRING)

Calling	Parameter	Description
---------	-----------	-------------

	Pal	String of 768 characters to store the palette in
--	-----	--

Returns None.

Description Gets the current palette and stores it into the Pal string. Such a palette can be then used by the **DQBsetPal**, **DQBfadeIn**, and **DQBsaveImage** functions.

Notes See Appendix A for information on how the palette is stored in the string.

Example See **DQBfadeTo** example.

DQBgline Sub

Prototype DECLARE SUB DQBgline (Layer AS INTEGER, x1 AS INTEGER, y1 AS _
INTEGER, x2 AS INTEGER, y2 AS INTEGER, Col1 AS INTEGER, Col2 AS _
INTEGER)

Calling	Parameter	Description
---------	-----------	-------------

	Layer	Layer to draw the line on
	x1	First endpoint x coordinate
	y1	First endpoint y coordinate
	x2	Second endpoint x coordinate
	y2	Second endpoint y coordinate
	Col1	First endpoint color
	Col2	Second endpoint color

Returns None.

Description Like **DQBline**, this function draws a line on the specified layer, but it interpolates colors between the two indexes specified at the endpoints. It is recommended that you call this function only after setting up a proper gradient palette.

Notes None.

Example None.

DQBgtri Sub

Prototype DECLARE SUB DQBgtri (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL c1, _
BYVAL x2, BYVAL y2, BYVAL c2, BYVAL x3, BYVAL y3, BYVAL c3)

Calling	Parameter	Description
	Layer	Layer to draw the triangle on
	x1	x coordinate of the first vertex
	y1	y coordinate of the first vertex
	c1	color of the first vertex
	x2	x coordinate of the second vertex
	y2	y coordinate of the second vertex
	c2	color of the second vertex
	x3	x coordinate of the third vertex
	y3	y coordinate of the third vertex
	c3	color of the third vertex

Returns None.

Description Draws a gouraud-shaded triangle with (x1, y1), (x2, y2), and (x3, y3) as vertices, interpolating specified colors, using the current palette. **DQBgtri** does not support transparency, but it's affected by the clipping box.

Notes Gouraud shading works better when you set a palette with lots of shades of the same color. It is up to you to find the best settings for your needs. See also **DQBtri**, **DQBbtri**, and **DQBttri**.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Set up the palette with shades of red
FOR i = 0 TO 63
    DQBsetCol i, i, 0, 0
    DQBsetCol 64 + i, 63, i, i
NEXT i

' Draw 500 gouraud-shaded triangles; uses only the first 128 colors
we set
FOR i = 1 TO 500
    DQBgtri VIDEO, (RND * 320), (RND * 200), (RND * 128), (RND _
320), (RND * 200), (RND * 128), (RND * 320), (RND * 200), (RND * 128)
NEXT i

' Wait for the user to press a key
```

```

WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBhPut Sub

Prototype DECLARE SUB DQBhPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff, BYVAL Col)

Calling	Parameter	Description
	Layer	Layer to draw the sprite on
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)
	Col	Drawing color

Returns None.

Description This function works exactly like normal **DQBput**, but it draws the whole sprite using the same specified color, disregarding the original colors. It supports clipping, but doesn't allow solid put mode. Color 0 is always treated as the transparent color.

Notes See also **DQBput**.

Example None.

DQBid\$ Function

Prototype DECLARE FUNCTION DQBid\$ ()

Calling None.

Returns A string containing the library version and the author's name.

Description Returns a small string that contains the library version and my name, in case you want to display them somewhere in your credits section.

Notes You can use **DQBver** to retrieve the library version only.

Example See **DQBver** example.

DQBinit Function

Prototype DECLARE FUNCTION DQBinit (BYVAL NumLayers, BYVAL NumSounds, BYVAL _
MemSize)

Calling	Parameter	Description
	NumLayers	Number of layers to allocate memory for
	NumSounds	Number of sounds to allocate memory for
	MemSize	Kilobytes (KB) of EMS memory to reserve for other program use

Returns	Integer	Description
	0	Initialization successful
	1	386 or better CPU not detected
	2	Unable to find an expanded memory manager
	3	Not enough free EMS memory to allocate specified number of layers
	4	Library has already been initialized

Description The **DQBinit** function must be called before any of the other library functions. It allocates the specified number of layers, sounds, and user memory in EMS, and does other initializations. Remember that you can allocate as many layers as you want, and up to 256 sounds, as long as enough free memory is available. If you do not have enough free EMS memory, you'll get an error 3 when calling this function. The same goes for the amount of EMS for other program use: the limit is the amount of free EMS available on your system.

The only thing you need to know is that each layer or sound takes 64 KB of EMS. Also, please keep in mind that if you refer to a layer or a sound number that has not been allocated with this procedure, your system will probably crash. For example, if you allocate 3 layers with **DQBinit**, you must not refer to layer 4. If you get an error 2, please refer to Chapter 1, "Setting Up EMM386.EXE", which explains how to enable EMS support on your system.

Notes As **DQBinit** must be called at the beginning of your program, remember to call **DQBclose** at its end. Initialized layers are automatically cleared (filled with pixel color 0), and the sounds are reset to silence and 0 length.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with four layers, no sounds, and no user EMS
Result = DQBinit(4, 0, 0)

' Close DirectQB
DQBclose
```

```
' A smarter way to handle errors would be to use DQError$, but for
' clarity here we'll do it this way:
SELECT CASE Result
CASE 0
    PRINT "Initialization successful"
CASE 1
    PRINT "Error: 386 or better CPU not detected!"
CASE 2
    PRINT "Error: unable to find an expanded memory manager!"
CASE 3
    PRINT "Error: not enough free EMS memory!"
END SELECT

' End the program
END
```

DQBinitText Sub

Prototype	DECLARE SUB DQBinitText()
Calling	None.
Returns	None.
Description	Initializes 80x25 standard text mode (BIOS mode 03h).
Notes	By using this sub instead of SCREEN 0 , you can obtain smaller executables once you compile your programs. See also DQBinitVGA .
Example	None.

DQBinitVGA Sub

Prototype	DECLARE SUB DQBinitVGA()
Calling	None.
Returns	None.
Description	Initializes 320x200 256-color VGA video mode (BIOS mode 013h), and resets the mouse if it was found.
Notes	By using this sub instead of SCREEN 13 , you can obtain smaller executables once you compile your programs. See also DQBinitText .
Example	None.

DQBinkey\$ Function

Prototype	DECLARE FUNCTION DQBinkey\$ ()
Calling	None.
Returns	If a key is pressed, it returns its character, otherwise a null string.
Description	DQBinkey\$ works just like the usual INKEY\$ QB statement. If any key is being pressed, this function returns its associated character; otherwise it returns a null string.
Notes	This function works only while the custom keyboard handler is on, otherwise you'll probably crash your system. In addition, as DQBinkey\$ does not get the character from the standard keyboard buffer, you'll lose your character if you do not store it in a variable.
Example	<pre> ' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with no layers, no sounds, and no user EMS IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror\$: END ' Install the keyboard handler DQBinstallKeyboard ' Prompt the user for a question PRINT "Press 'Y' if you like DirectQB, otherwise 'N'" DO DO: k\$ = DQBinkey\$: LOOP WHILE k\$ = "" SELECT CASE k\$ CASE "Y": PRINT "Thank you!": EXIT DO CASE "N": PRINT "Too bad!!": EXIT DO LOOP ' Wait for the user to press a key DQBwaitKey KEYANY ' Remove the keyboard handler DQBremoveKeyboard ' End the program DQBclose END </pre>

DQBinstallKeyboard Sub

Prototype DECLARE SUB DQBinstallKeyboard ()

Calling None.

Returns None.

Description Installs a custom keyboard interrupt handler that allows you to detect the exact state (pressed or released) of any key at any time. Once the interrupt handler is activated, the standard QB routines like **INPUT**, **INKEY\$**, or any other routine that uses the keyboard as input, will be unavailable. Calling them will probably result in your machine crashing, so take care. These standard routines can be called only when the keyboard interrupt handler is uninstalled by calling **DQBremoveKeyboard**. **DQBclose** automatically calls **DQBremoveKeyboard**, so you don't have to worry about it. When the handler is active, you can check the state of each key by simply calling the **DQBkey** function.

Notes None.

Example See **DQBkey** example.

DQBinstallSB Function

Prototype DECLARE FUNCTION DQBinstallSB (BYVAL VolActive, BYVAL Voices, BYVAL _
Freq, BYVAL BaseAddr, BYVAL IRQ, BYVAL DMA)

Calling	Parameter	Description
	VolActive	True to use 16K base memory for the voices volume table
	Voices	Number of voices to mix at real-time
	Freq	Sampling rate
	BaseAddr	Base address of your sound card
	IRQ	IRQ number setting
	DMA	Low (8-bit) DMA channel to be used

Returns	Integer	Description
	0	Initialization successful
	1	No sounds were allocated by DQBinit
	2	Soundcard not found or DSP failed to reset
	3	Old soundcard not supported
	4	Specified DMA channel is not supported
	5	Autodetection failed as the BLASTER variable is not set
	6	High mixing speed not supported
	7	Not enough memory to create the volume table

Description **DQBinstallSB** installs a custom IRQ routine and starts the sound engine. If you don't know your sound settings (base address, IRQ, or DMA), you can pass the AUTO constant as the missing parameter(s). This will force the function to search the BLASTER environment variable (if available) for the right setting. The number of voices can range from 1 up to 32, and tells the sound engine how many digital sound channels to mix at real-time. Needless to say, on older computers, too many voices will slow down your programs.

The sampling rate may range from 3892 Hz to 23000 Hz. If you specify true as the VolActive parameter, you'll need 16K of base memory, as the function will build an internal volume table. This will allow you to set custom volume settings to each of the initialized voices. If you don't need to play sounds with different volumes at the same time, you don't need this table, so just pass false for the parameter.

Notes As always, the **SETMEM** function should be used to free memory from QB if you need base memory for the volume table. To turn the sound engine off, call **DQBremoveSB**. **DQBclose** automatically does this for you. See also **DQBloadSound**, **DQBplaySound**, and **DQBstopVoice**.

Example See **DQBplaySound** example.

DQBinUse Function

Prototype	DECLARE FUNCTION DQBinUse (BYVAL Voice)	
Calling	Parameter	Description
	Voice	Voice number to retrieve the status of
Returns	True if a sound is currently being played on specified voice.	
Description	DQBinUse returns true if any sound is currently playing on the specified voice, otherwise false.	
Notes	None.	

Example None.

DQBjoyDetected Function

Prototype DECLARE FUNCTION DQBjoyDetected (BYVAL JoyNum)

Calling	Parameter	Description
---------	-----------	-------------

	JoyNum	Joystick number to detect
--	--------	---------------------------

Returns True if specified joystick has been detected, otherwise false.

Description This function should be called before using the other joystick functions, to detect if a joystick is available. The parameter JoyNum can be 0 for joystick 1, 1 for joystick 2, or 2 for 4-button joystick. If you can't remember this, just call this function, as well as the other joystick functions, with the constants JOY1, JOY2, and GAMEPAD defined in the file DIRECTQB.BI), and explained in Appendix A.

Notes None.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Find if joystick 1 is available
IF DQBjoyDetected(JOY1) THEN
    PRINT "Joystick 1 detected"
ELSE
    PRINT "Joystick 1 not detected"
END IF

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END
```

DQBjoyFire Function

Prototype DECLARE FUNCTION DQBjoyFire (BYVAL JoyNum, BYVAL Button)

Calling	Parameter	Description
	JoyNum Button	Joystick number to retrieve status of Button number to check
Returns	True if specified button of specified joystick is pressed, otherwise false.	
Description	This function is used to check if a button of the specified joystick is being pressed. For information about the JoyNum parameter, please refer to the DQBjoyDetected function. The other parameter, Button, can be 0 for button A, or 1 for button B. If you're handling a 4-button joystick, you can also use 2 and 3 for buttons C and D. The buttons also have constants named BUTA, BUTB, BUTC, and BUTD (defined in the file DIRECTQB.BI), and explained in Appendix A.	
Notes	None.	
Example	See DQBjoyMove example.	

DQBjoyMove Function

Prototype DECLARE FUNCTION DQBjoyMove (BYVAL JoyNum, BYVAL Direction)

Calling	Parameter	Description
	JoyNum Direction	Joystick number to retrieve status of Direction to check (see below)
Returns	True if the joystick is moved in the given direction, otherwise false.	
Description	Call this function to check which direction the joystick has moved. For information about the JoyNum parameter, refer to the DQBjoyDetected function. The Direction parameter is an integer that must be set to:	
	Integer	Direction
	0	Up
	1	Down
	2	Left
	3	Right

There are, of course, four constants for this parameter, named UP, DOWN, LEFT, and RIGHT, (defined in the file DIRECTQB.BI) explained in Appendix A.

Notes None.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'
DIM X, Y, Col

' Initialize the library with one layer, no sounds, and no user EMS
IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

IF NOT DQBjoyDetected(JOY1) THEN
    PRINT "Joystick 1 not detected, program aborted."
    DQBclose
    END
END IF

DQBinitVGA

X = 150: Y = 90

DO
    ' Poll joystick 1
    DQBpollJoy JOY1

    ' Check movements
    IF DQBjoyMove(JOY1, UP) THEN Y = Y - 1      ' Joystick 1 up

    IF DQBjoyMove(JOY1, DOWN) THEN Y = Y + 1    ' Joystick 1 down

    IF DQBjoyMove(JOY1, LEFT) THEN X = X - 1    ' Joystick 1 left

    IF DQBjoyMove(JOY1, RIGHT) THEN X = X + 1   ' Joystick 1 right

    ' Adjust coordinates
    IF X < 0 THEN X = 0
    IF X > 299 THEN X = 299
    IF Y < 0 THEN Y = 0
    IF Y > 179 THEN Y = 179

    ' Reset box color to white
    Col = 15

    ' If button A is pressed, change color to red
    IF DQBjoyFire(JOY1, BUTA) THEN Col = 40

    ' If button B is pressed, change color to green
    IF DQBjoyFire(JOY1, BUTB) THEN Col = 2

    ' Clear the contents of layer 1
    DQBclearLayer 1
    ' Draw the box
    DQBboxf 1, X, Y, (X + 19), (Y + 19), Col

    ' Wait for vertical retrace 1 time
    DQBwait 1
```

```

' Copy our layer to the screen
DQBcopyLayer 1, VIDEO

' Print a message
LOCATE 1
PRINT " Joystick demo - press a key to exit!"

LOOP WHILE INKEY$ = ""

' End the program
DQBclose
END

```

DQBjoyX Function

Prototype DECLARE FUNCTION DQBjoyX (BYVAL JoyNum)

Calling	Parameter	Description
---------	-----------	-------------

JoyNum	Joystick number to return x-axis position of
--------	--

Returns An integer value holding the current joystick x-axis position.

Description This function can be used to retrieve the exact x-axis position of specified joystick, relative to the center position. The center position is automatically detected at startup, or can be set by calling **DQBresetJoy**.

Notes None.

Example None.

DQBjoyY Function

Prototype DECLARE FUNCTION DQBjoyY (BYVAL JoyNum)

Calling	Parameter	Description
---------	-----------	-------------

JoyNum	Joystick number to return y-axis position of
--------	--

Returns An integer value holding the current joystick y-axis position.

Description This function can be used to retrieve the exact y-axis position of specified joystick, relative to the center position. The center position is automatically detected at startup, or can be set by calling **DQBresetJoy**.

Notes None.

Example None.

DQBkey Function

Prototype DECLARE FUNCTION DQBkey (BYVAL ScanCode)

Calling	Parameter	Description
---------	-----------	-------------

	ScanCode	Scancode of the key to retrieve status of
--	----------	---

Returns True if the key is pressed, otherwise false.

Description This function returns useful information only when the custom keyboard interrupt handler has been installed by calling **DQBinstallKeyboard**. For a complete list of keyboard scancodes, refer to Appendix B. There are also a few constants declared into the file DIRECTQB.BI, such as KEYESC, KEYENTER, KEYUP, KEYDOWN, KEYLEFT, KEYRIGHT, and others, explained in Appendix A.

Notes None.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Install the custom keyboard handler
DQBinstallKeyboard

CLS
PRINT "Keyboard demo - press any key to see its scancode. Multiple _
keypresses are allowed; press ESC to exit."

DO
    LOCATE 3, 1

    ' Scan every possible key
    FOR i = 0 TO 127
        ' If the key is pressed, prints its scancode
        IF DQBkey(i) THEN PRINT "[" + LTRIM$(STR$(i)) + "]" ";
    NEXT i

    ' Clear a portion of the screen
    PRINT SPACE$(40)

    ' Loop while the ESC key is not pressed
    LOOP UNTIL DQBkey(KEYESC)

    ' Uninstall the keyboard handler (this is not really needed when
    ' we're going to end the program by calling DQBclose)
    DQBremoveKeyboard
```

```
' End the program
DQBclose
END
```

DQBlen Function

Prototype DECLARE FUNCTION DQBlen (Text AS STRING)

Calling **Parameter** **Description**

Text Text string to retrieve the length of

Returns An integer holding the length in pixels of the given string, assuming the current font character length.

Description Like **LEN**, **DQBlen** returns the length of a specified string, but in pixel units. Because DirectQB supports non-fixed sized fonts, this function is useful to retrieve the length of a string in pixels, calculated using the length of the characters of the current font.

Notes None.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

Length = DQBlen("The length of this string is:")
DQBprint VIDEO, "The length of this string is:" + STR$(Length) _
+ " pixels", 0, 0, 31

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END
```

DQBline Sub

Prototype DECLARE SUB DQBline (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, _
BYVAL y2, BYVAL Col)

Calling **Parameter** **Description**

Layer	Layer to draw the line on
x1	First point x coordinate
y1	First point y coordinate
x2	Second point x coordinate
y2	Second point y coordinate
Col	Drawing color

Returns None.

Description This procedure draws a line from the point at (x1, y1) to the point at (x2, y2), on the given layer and with the specified color. The Bresenham's line algorithm is used.

Notes **DQBline** is affected by the clipping box. See also **DQBgline**.

Example None.

DQBloadBMap Function

Prototype DECLARE FUNCTION DQBloadBMap (BMap AS INTEGER, FileName AS STRING)

Calling **Parameter** **Description**

BMap	Number of the blender map to load data into
FileName	Blender map filename with full extension

Returns **Integer** **Description**

0	Loading successful
1	Blender map not yet created
2	Cannot open file, or file does not exist
3	Bad or unknown file format
4	Incompatible blender map
5	General reading error

Description Loads data from the specified binary file into the specified blender map. The blender map must have been previously created by calling **DQBcreateBMap**.

Notes The blender map you want to load data into must be compatible with the map saved in the file. This means that the number of mapped foreground colors must be the same for both the map stored in the file and the map structure you are about to load it into. See also **DQBcreateBMap** and **DQBsaveBMap**.

Example None.

DQBloadFont Function

Prototype DECLARE FUNCTION DQBloadFont (FileName AS STRING)

Calling	Parameter	Description
	FileName	Font file with full extension
Returns	Integer	Description
	0	Loading successful
	1	Cannot open file, or file does not exist
	2	General reading error

Description Loads and sets the font from the specified file. The font file must contain the data in the first 2305 bytes, in the format explained in Appendix C.

Notes This function is useful because it doesn't require a QB buffer to hold the data. The disadvantage of using it is that every time you change the font, the disk is accessed. By using **DQBsetFont** instead, you can keep your fonts in QB memory and set them as needed.

Example None.

DQBloadImage Function

Prototype DECLARE FUNCTION DQBloadImage (Layer AS INTEGER, x AS INTEGER, y AS _
INTEGER, FileName AS STRING, Pal AS STRING, ImgWidth AS INTEGER, _
ImgHeight AS INTEGER)

Calling	Parameter	Description
	Layer	Layer to load the image onto
	x	x position where to begin drawing
	y	y position where to begin drawing
	FileName	Image filename with full extension
	Pal	String where the image palette is loaded into
	ImgWidth	Integer variable to store the image width
	ImgHeight	Integer variable to store the image height
Returns	Integer	Description
	0	Loading successful
	1	Cannot open file, or file does not exist
	2	General reading error
	3	Bad or unknown file format
Description	Loads an image into the specified layer, at the specified coordinates. Images can be any size, up to 320x200 pixels, but must be saved with 256 colors. You can load bitmap (BMP), PC Paintbrush (PCX), or BSAVE (BSV) files; the format is automatically detected and is independent from the file extension. The function returns 0 if the operation is successful, otherwise an error code is returned (see above). When loading a BMP or a PCX image, the palette is stored in the Pal string. The current palette is not changed, allowing you to change it only when you want (i.e. by calling DQBsetPal or DQBfadeIn). If you're loading a BSV image, the function searches for the palette appended at the end of the file. If it exists, the palette will be stored in the Pal string as well. In all cases, the image width and height will be stored in the ImgWidth and ImgHeight variables.	
Notes	See Appendix C for the palette string format.	
Example	None.	

DQBloadRawSound Function

Prototype DECLARE FUNCTION DQBloadRawSound (Slot AS INTEGER, FileName AS _
STRING, Offset AS LONG, Length AS INTEGER)

Calling	Parameter	Description
	Slot	Sound slot to load the sound into
	FileName	Sound filename with full extension
	Offset	File position where to begin to load in bytes
	Length	Sound data length in bytes

Returns	Integer	Description
	0	Loading successful
	1	Cannot open file, or file does not exist
	2	General reading error
	3	Cannot read past the end of file
	4	Specified sound slot does not exist
Description	Loads sound data from a specified position in a given file. This function directly stores the sound in the specified slot, and does not check for the data format. DQBLoadRawSound can be useful when handling sound data files holding multiple sounds, as well as when DQBLoadSound fails the sound data format check.	
Notes	None.	
Example	None.	

DQBLoadSound Function

Prototype DECLARE FUNCTION DQBLoadSound (Slot AS INTEGER, FileName AS STRING)

Calling	Parameter	Description
	Slot	Sound slot to load sound into
	FileName	Sample filename with full extension
Returns	Integer	Description
	0	Loading successful
	1	Cannot open file, or file does not exist
	2	General reading error
	3	Bad or unknown file format
	4	Sound format not yet supported
	5	Sound file exceeds the 64K length limit
	6	Specified sound slot does not exist
Description	Loads the sound from the file and stores it in EMS. The Slot parameter identifies the sound effect and is used by the DQBplaySound function.	
Notes	Only 8-bit mono WAV files are supported.	
Example	See DQBplaySound example.	

DQBmapLayer Function

Prototype DECLARE FUNCTION DQBmapLayer (BYVAL Layer)

Calling **Parameter** **Description**

Parameter	Description
Layer	Number of the layer to map

Returns An integer value holding the segment where you can access the layer data.

Description This function is useful when you want to have complete control over a specified layer. In fact, **DQBmapLayer** returns the segment of the 64K memory window where the layer data is held. If the layer is in EMS, the function returns the EMS pageframe, and it also maps the specified layer to the pageframe.

Notes None.

Example None.

DQBmouseDetected Function

Prototype DECLARE FUNCTION DQBmouseDetected ()

Calling None.

Returns **Integer** **Description**

Integer	Description
0	Mouse not detected
-1	Mouse detected and successfully initialized

Description Returns true if a mouse has been detected on your system. You should call this before calling any other mouse routine.

Notes It is recommended that you also call **DQBresetMouse** just after setting mode 13h by calling **DQBinitVGA**, before using any other mouse routine.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with one layer, no sounds, and no user EMS
IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Check if a mouse has been detected
IF NOT DQBmouseDetected THEN
    ' Mouse has not been detected
```

```

        PRINT "This demo requires a mouse to run!"
        DQBclose
    END
END IF

DQBinitVGA

' Reset the mouse cursor and range
DQBresetMouse

' Set a new mouse range
DQBsetMouseRange 6, 6, 314, 148

DO
    ' Clear layer 1
    DQBclearLayer 1

    DQBbox 1, 5, 5, 315, 149, 32

    ' Print some stats
    DQBprint 1, "PRESS ANY KEY TO EXIT DEMO", 0, 170, 40

    ' Prepare our info string
    Info$ = "Mouse is at" + str$(DQBmouseX) + "," + str$(DQBmouseY)
    Info$ = Info$ + " - Buttons: " + str$(DQBmouseLB) + " " + _
    str$(DQBmouseRB)

    DQBprint 1, Info$, 0, 180, 40

    ' Hide the mouse cursor before drawing to the screen
    DQBmouseHide

    ' Copy layer 1 onto the screen
    DQBcopyLayer 1, VIDEO

    ' Show the cursor again
    DQBmouseShow

    ' Loop while no keys are pressed
    LOOP WHILE INKEY$ = ""

    ' End the program
    DQBclose
END

```

DQBmouseHide Sub

Prototype DECLARE SUB DQBmouseHide ()

Calling None.

Returns None.

Description Hides the mouse cursor if it was visible. Even when the mouse is not visible, it remains active.

Notes See **DQBmouseShow**.

Example See **DQBmouseDetected** example.

DQBmouseLB Function

Prototype DECLARE FUNCTION DQBmouseLB ()

Calling None.

Returns	Integer	Description
----------------	----------------	--------------------

0	Released
---	----------

-1	Pressed
----	---------

Description Call this function to check if the left mouse button is pressed or released. This function can be called at any time, as the mouse status is automatically updated.

Notes None.

Example See **DQBmouseDetected** example.

DQBmouseRB Function

Prototype DECLARE FUNCTION DQBmouseRB ()

Calling None.

Returns	Integer	Description
----------------	----------------	--------------------

0	Released
---	----------

-1	Pressed
----	---------

Description Call this function to check if the right mouse button is pressed or released. This function can be called at any time, as the mouse status is automatically updated.

Notes None.

Example See **DQBmouseDetected** example.

DQBmouseShow Sub

Prototype	DECLARE SUB DQBmouseShow ()
Calling	None.
Returns	None.
Description	Shows the mouse cursor if it was not visible.
Notes	None.
Example	See DQBmouseDetected example.

DQBmouseX Function

Prototype	DECLARE FUNCTION DQBmouseX ()
Calling	None
Returns	An integer value holding the current mouse x coordinate
Description	Call this function at any time to check the current mouse x position.
Notes	The position is given in pixel units, and ranges depending on the actual mouse range box. The mouse range box can be changed by calling DQBsetMouseRange .
Example	See DQBmouseDetected example.

DQBmouseY Function

Prototype	DECLARE FUNCTION DQBmouseY ()
Calling	None.
Returns	An integer value holding the current mouse y coordinate
Description	Call this function at any time to check the current mouse y position.
Notes	The position is given in pixel units, and ranges depending on the actual mouse range box. The mouse range box can be changed by calling DQBsetMouseRange .
Example	See DQBmouseDetected example.

DQBmPut Sub

Prototype DECLARE SUB DQBmPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff, BYVAL Flip)

Calling	Parameter	Description
	Layer	Layer where to draw the sprite
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)
	Flip	Flip flag

Returns None.

Description This function works almost like **DQBput**, but it can draw sprites flipped horizontally, vertically, or both. The flip parameter tells **DQBmPut** how to flip the sprite:

Flip Flag	Description
0	No flipping
1	Horizontal flipping
2	Vertical flipping
3	Both horizontal and vertical flipping (1 + 2 = 3)

You can use the constants defined in **DIRECTQB.BI** to simplify your calls; see Appendix A for details.

Notes **DQBmPut** supports transparency and clipping. See also **DQBput**.

Example None.

DQBnumDrives Function

Prototype DECLARE FUNCTION DQBnumDrives ()

Calling None.

Returns An integer value holding the number of drives.

Description This function returns the number of available logical drives. For example, a value of 3 the drives A: B: and C: are present.

Notes Drive B: (secondary floppy) may be a “ghost” drive on some systems. See also **DQBdir\$**, **DQBpath\$**, **DQBdrive\$**, and **DQBsetDrive**.

Example See **DQBdir\$** example.

DQBopenDataFile Function

Prototype DECLARE SUB DQBopenDataFile (FileName AS STRING, Password AS _
STRING)

Calling	Parameter	Description
	FileName	Datafile to open
	Password	Password for data decoding

Returns	Integer	Description
	0	Operation successful
	1	Cannot open file or file does not exist
	2	General file reading error
	3	Bad datafile format
	4	Can't open two datafiles at once

Description Opens a datafile and sets the decoding password. The datafile must have been created by the DirectQB DataFile Encoder program, or any other program able to generate DirectQB-compatible datafiles. The password can be any string up to 8 characters.

Notes You can pass a null string as a password, meaning the datafile is not password-protected. You cannot open two datafiles at once. If you need to access data that is stored in another file, you must first close the open datafile with the **DQBcloseDataFile** function. See also **DQBunpackImage**, **DQBunpackSprite**, **DQBunpackSound**, **DQBunpackPal**, **DQBunpackBMap**, **DQBunpackFont**, **DQBunpackCursor**, **DQBunpackUser**, and Appendix C for the datafile file format.

Example

```
' This example assumes there is a datafile named TEST.BIN, containing
' an image whose packet ID is SAMPLE (stored in TEST.BI).

' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Here we include the BI file generated by DQBENC for our datafile
'$INCLUDE: 'TEST.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Opens datafile TEST.BIN, using password PASS
```

```

IF DQBopenDataFile("TEST.BIN","PASS") THEN DQBclose: PRINT _
DQLError$: END

' Setup VGA
DQBinitVGA

' Unpacks on the screen from TEST.BIN the image whose ID is SAMPLE,
' placing it at coordinates (0,0)
IF DQBunpackImage(SAMPLE, VIDEO, 0, 0) THEN DQBclose: PRINT _
DQLError$: END

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' Close datafile (not really needed here, as DQBclose does it
' automatically)
DQBcloseDataFile

' End the program
DQBclose
END

```

DQBopenFLI Function

Prototype DECLARE SUB DQBopenFLI (FileName AS STRING, Frames AS INTEGER, _
Speed AS INTEGER)

Calling	Parameter	Description
	FileName	FLI animation filename
	Frames	Integer variable that will hold the total frames number
	Speed	Integer variable that will hold the animation speed factor

Returns	Integer	Description
	0	Operation successful
	1	Not enough free base memory
	2	Cannot open file or file does not exist
	3	General file reading error
	4	Bad or unknown file format
	5	A FLI file has already been opened

Description Opens a FLI file and initializes the animation. The DirectQB internal FLI routines cannot handle more than one file opened at the same time. If you call this function and then you try to call it again, or to call **DQBplayFLI**, you'll get error 5. An animation file can be closed by using the **DQBcloseFLI** function. **DQBopenFLI** also allows you to get the number of frames and the speed of the specified FLI file, by passing QB variables to the function.

Notes Like **DQBplayFLI**, this function also requires 64K of additional base memory as a decoding buffer. See also **DQBcloseFLI**, **DQBplayFLIstep**, and **DQBplayFLI**.

Example See **DQBplayFLIstep** example.

DQBpaint Sub

Prototype DECLARE SUB DQBpaint (BYVAL Layer, BYVAL x, BYVAL y, BYVAL Col)

Calling	Parameter	Description
---------	-----------	-------------

Layer	Layer to paint on
x	x coordinate where to start filling
y	y coordinate where to start filling
Col	Filling color

Returns None.

Description Similar to QB's **PAINT** statement, this function fills any closed area, starting with the specified coordinates. The area is filled until a border with a color different from specified one is encountered.

Notes This function is actually slower than **PAINT**, but **DQBpaint** has the advantage that it fills only until a border with any different color is hit.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Draw a closed figure on the screen
DQBellipse VIDEO, 160, 100, 100, 60, 40
DQBbox VIDEO, 100, 30, 150, 110, 11

' Paint it!
DQBpaint VIDEO, 160, 100, 4

' Wait for the user to press a key...
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END
```

DQBpalOff Sub

Prototype	DECLARE SUB DQBpalOff ()
Calling	None.
Returns	None.
Description	By calling this procedure, all the colors in the current palette are set to black. This can be used to “turn off” the screen when preparing it.
Notes	None.
Example	None.

DQBpalRotate Sub

Prototype	DECLARE SUB DQBpalRotate (BYVAL FirstCol, BYVAL LastCol, BYVAL _ RotDir)
------------------	---

Calling	Parameter	Description
	FirstCol	First color index of palette interval to rotate
	LastCol	Last color index of palette interval to rotate
	RotDir	Rotating direction flag

Returns	None.
----------------	-------

Description	Rotates the interval of the current palette specified by the FirstCol and LastCol parameters forward or backward.
--------------------	---

RotDir	Description
0	Palette is rotated forward
<i>Anything else</i>	Palette is rotated backward

You can also use the FORWARD or BACKWARD constants (defined in DIRECTQB.BI), as explained in Appendix A

Notes	LastCol must be greater or equal than FirstCol. See also DQBsetCol , DQBgetCol , DQBsetPal , and DQBgetPal .
--------------	--

Example	None.
----------------	-------

DQBpath\$ Function

Prototype	DECLARE FUNCTION DQBpath\$ ()
Calling	None.
Returns	A string holding the current drive path.
Description	DQBpath\$ returns the actual full path of the current drive. Windows long directory names are supported.
Notes	See also DQBdir\$, DQBdrive\$, DQBnumDrives , DQBsetDrive .
Example	See DQBdir\$ example.

DQBpauseSound Sub

Prototype	DECLARE SUB DQBpauseSound ()
Calling	None.
Returns	None.
Description	Pauses voices sound output.
Notes	See also DQBresumeSound .
Example	None.

DQBpeek Sub

Prototype	DECLARE SUB DQBpeek (BYVAL DataSeg, BYVAL DataOff, BYVAL Offset AS _ LONG, BYVAL Length)	
Calling	Parameter	Description
	DataSeg	Segment of QB array or variable where data is copied from EMS
	DataOff	Offset of QB array or variable where data is copied from EMS
	Offset	Absolute offset into allocated EMS memory where to take data
	Length	Length in bytes to copy
Returns	None.	

Description	DQBpeek copies data from the user EMS memory area (this is set by the third parameter of the DQBinit function) into a QB array or variable. The Offset parameter specifies (in byte units) where to begin copying data. This offset is absolute, and this means you refer to your user EMS memory pool as a flat table. Length bytes are copied from EMS.
Notes	This function works best with large amount of data. See also DQBpoke .
Example	See DQBpoke example.

DQBplayFLI Function

Prototype DECLARE FUNCTION DQBplayFLI (FileName AS STRING, BufLayer AS _
INTEGER, KeyStop AS INTEGER, LoopFlag AS INTEGER)

Calling	Parameter	Description
	FileName	FLI filename
	BufLayer	Layer to be used for double buffering
	KeyStop	Scancode of the stop key
	LoopFlag	Flag for animation looping
Returns	Integer	Description
	0	Operation successful
	1	Not enough free base memory
	2	Cannot open file or file does not exist
	3	General file reading error
	4	Bad or unknown file format

Description Plays a whole animation from the specified FLI file. FLC files are not currently supported. The file can be played using a double buffer, to achieve flickerless animations. This feature requires an additional existing layer number to be specified in the BufLayer parameter; this layer can be in EMS as well as in base memory. If you tell the function to use VIDEO as the needed layer, the animation will be decoded directly onto the screen, avoiding double buffering. On fast computers, this will look almost the same as if the buffer is used, but on older machines it is recommended that you always use buffered decoding.

The animation can be looped, and you can also specify a key to stop it at any time and return the execution to your program. By specifying KEYANY as the KeyStop parameter, the player will stop once any key is pressed. If you don't want to allow the user to stop an animation, pass NONE as this parameter. The use of the stop key feature requires that the DirectQB keyboard handler has been previously installed; otherwise the function will ignore any keyboard strokes. The last parameter tells **DQBplayFLI** whether or not to loop the animation once the last frame is reached;

you can use the ONCE and LOOPED constants here. To avoid infinite loops when playing a looped animation, a stop key must be specified, and therefore you must first install the keyboard handler. If no stop key is defined, and you try to play an animation with looping, it will be played as if the ONCE mode was selected.

Notes

Other than the layer used for double buffering, this function also requires 64K of free base memory for fast decoding purposes. As always, you should call **SETMEM** to free some memory from the QB far heap, so that DirectQB functions can use it. If you're also using a blender map, and you don't have much free base memory, you may want to call **DQBremoveBMap** before calling **DQBplayFLI**. After the animation ends, you can reallocate the blender map; this way only 64K base memory is needed. If you want to have more control over the animation (like adding text over each frame before showing it, or changing the playback speed), you can use the **DQBopenFLI**, **DQBplayFLIstep**, and **DQBcloseFLI** functions. Check them out!

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with one layer, no sounds, and no user EMS
IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Prompt the user for the FLI filename
INPUT "Enter full FLI filename (hit enter to quit): ", FileName$

' Switch to VGA mode
DQBinitVGA

' Install the keyboard handler
DQBinstallKeyboard

' Free 64K of memory for animation decoding
AllocMem& = SETMEM(-65537)

IF FileName$ <> "" THEN

    ' Play looped animation, using layer 1 for double buffering
    IF DQBplayFLI(FileName$, 1, KEYANY, LOOPED) THEN
        ' An error has occurred
        DQBclose
        PRINT DQBerror$
        FreeMem& = SETMEM(65537)
        END

    END IF

END IF

' Stop the animation whenever the user press any key
DQBclose
FreeMem& = SETMEM(65537)
END
```

DQBplayFLIstep Sub

Prototype DECLARE SUB DQBplayFLIstep (BYVAL Layer)

Calling **Parameter** **Description**

Layer	Layer where the frame has to be drawn onto
-------	--

Returns None.

Description Decodes the next frame of a FLI file onto a specified layer. The animation must have been opened with **DQBopenFLI** before calling this function. When the last frame is reached, **DQBplayFLIstep** will automatically restart from the first one, so it's up to the programmer to stop the animation at the right time. It's also up to the programmer to add a proper delay before copying the frame onto the screen. By the way, **DQBopenFLI** returns both the total number of frames and the speed factor of the FLI file to be opened. The speed factor can be used as a parameter to the **DQBwait** function, to achieve the default animation speed.

Notes Of course, you can decode each frame directly onto the screen, but as explained in the description of the **DQBplayFLI** routine, this might not look so good on older computers. Also, keep in mind that by calling this function, the palette may change at any time as the FLI animation is playing. In addition, you should know that when a new frame is decoded onto a layer, only the parts of the frame that have changed from the last frame are drawn (that's the FLI standard). If you plan to draw some moving stuff over the animation, you should always decode each frame onto the same layer, copy this layer onto a new one, and then draw your own stuff on the last one before copying it onto the screen.

Example ' This program does the exact thing as the Example of DQBplayFLI, but
 ' this uses the three functions DQBopenFLI, DQBplayFLIstep and
 ' DQBcloseFLI to achieve the result.

```

' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Declare two variables to hold the number of frames and speed factor
DIM FramesNumber AS INTEGER, SpeedFactor AS INTEGER

' Initialize the library with two layers, no sounds, and no user EMS
IF DQBinit(2, 0, 0) THEN DQBclose: PRINT DQBerror$: END

' Prompt the user for the FLI filename
INPUT "Enter full FLI filename (hit enter to quit): ", FileName$

IF FileName$ = "" THEN DQBclose: END

' Frees 64K of memory for animation decoding
AllocMem& = SETMEM(-65537)
```

```

' Open the animation file and get information on it
IF DQBopenFLI(FileName$, FramesNumber, SpeedFactor) THEN
  ' An error has occurred
  DQBclose
  FreeMem& = SETMEM(65537)
  PRINT DQBerror$: END
END IF

' Show some stats to the user
PRINT FileName$ + " contains" + STR$(FramesNumber) + " frames and _
has a ";
PRINT "speed factor of" + STR$(SpeedFactor)
PRINT "Press a key to continue..."
WHILE INKEY$ = "": WEND

' Switch to VGA mode
DQBinitVGA

' Play the animation and automatically loop it
DO
  ' Decode next frame on layer 2
  DQBplayFLIstep 2

  ' Copy new frame into layer 1, so we can draw on it without
  ' problems
  DQBcopyLayer 2, 1

  ' Now the frame has been drawn onto layer 1; you can do your stuff
  ' here
  ' ...

  ' Add a delay and copy the frame onto the screen
  DQBwait SpeedFactor
  DQBcopyLayer 1, VIDEO
LOOP WHILE INKEY$ = ""

' Stop the animation whenever the user press any key
' Close the FLI file
DQBcloseFLI

' End the program
DQBclose
FreeMem& = SETMEM(65537)
END

```

DQBplaySound Sub

Prototype DECLARE SUB DQBplaySound (BYVAL Slot, BYVAL Voice, BYVAL Freq, _
BYVAL LoopFlag)

Calling	Parameter	Description
	Slot	Slot holding the sound data
	Voice	Sound channel to use
	Freq	Playing frequency
	LoopFlag	Specifies to play the sound once or repeatedly
Returns	None	
Description	Plays a sound effect previously stored in EMS by the DQBloadSound function in a specified slot. The voice parameter can range from 1 to the maximum number of voices specified when calling DQBinstallSB , and tells DQBplaySound which sound channel to use.	
	LoopFlag	Description
	0	Sound is played once
	1	Sound is looped
	There are two constants defined in DIRECTQB.BI that help you use the LoopFlag parameter. They are the ONCE and the LOOPED constants. The DirectQB sound engine supports sound resampling; just specify the output frequency for each sound you play, and the sound will be resampled and played in real-time.	
Notes	If you play a sound while another one is still playing on the same voice, the old sound is stopped and the new sound starts. To stop any sound playing on a voice, call DQBstopVoice .	
Example	<pre> ' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with no layers, one sound, and no user EMS IF DQBinit(0, 1, 0) THEN DQBclose: PRINT DQBerror\$: END ' Free 16K of base memory for the sound engine volume table AllocMem& = SETMEM(-16385) ' Initialize the sound engine, by passing 220h as the base address ' and by autodetecting the IRQ and DMA settings. 2 voices are ' initialized, the sampling rate is set to 22 KHz, and the volume ' table is built. IF DQBinstallSB(TRUE, 2, 22050, &H220, AUTO, AUTO) THEN ' Bad initialization DQBclose PRINT DQBerror\$ END END IF ' Ask the user for the location of a WAV file to load INPUT "Insert the WAV filename to load:", File\$ ' Load the sound in memory </pre>	

```

IF DQBloadSound(1, File$) THEN DQBclose: PRINT DQBerror$: END

PRINT "Press a key to exit..."

' Play sound 1 at 22 Khz on voice 2 once
DQBplaySound 1, 2, 22050, ONCE

' Wait for a key
WHILE INKEY$ = "": WEND

' Turn off the sound engine (that's not really needed, as it's
' automatically done for you by the DQBclose function)
DQBremoveSB

' End the program
DQBclose
FreeMem& = SETMEM(16385)
END

```

DQBpoint Function

Prototype DECLARE FUNCTION DQBpoint (BYVAL Layer, BYVAL x, BYVAL y)

Calling	Parameter	Description
	Layer	Layer to get the pixel from
	x	Pixel x coordinate
	y	Pixel y coordinate

Returns An integer value holding the color number of specified pixel.

Description Call this function to retrieve the color number of a specified pixel.

Notes None.

Example None.

DQBpoke Sub

Prototype DECLARE SUB DQBpoke (BYVAL DataSeg, BYVAL DataOff, BYVAL Offset AS _
LONG, BYVAL Length)

Calling	Parameter	Description
	DataSeg	Segment of QB array or variable where data to copy to EMS is
	DataOff	Offset of QB array or variable where data to copy to EMS is
	Offset	Absolute offset in allocated EMS memory to copy the data to
	Length	Length in bytes to copy
Returns	None.	
Description	DQBpoke is used to write data from a QB array or variable to the user EMS memory pool. You must specify the address and the length in bytes of the source data, plus the EMS offset to copy the data to. As with DQBpeek , this address is absolute, so refer to your user EMS area as a flat memory area.	
Notes	Never try to copy data when it (or a part of it) will lie outside the user EMS memory area set by the DQBinit function; you may end up with a machine crash. See also DQBpeek .	
Example	<pre>' All integers for speed DEFINT A-Z '\$INCLUDE: 'directqb.bi' ' Initialize the library with no layers, no sounds, and 100K user ' EMS IF DQBinit(0, 0, 100) THEN DQBclose: PRINT DQBerror\$: END ' Here we declare a string we'll copy to EMS DIM Text AS STRING * 30 Text = "This is a test!!" ' Display our string PRINT "Now the string is: " + Text ' Copy the string to EMS, at address 0 of our user memory area DQBpoke VARSEG(Text), VARPTR(Text), 0, 30 ' Clear our string and display it Text = "" PRINT "Now the string is: " + Text ' Get the string back from the EMS user memory area DQBpeek VARSEG(Text), VARPTR(Text), 0, 30 ' Display it again PRINT "Now the string is: " + Text ' End the program DQBclose END</pre>	

DQBpollJoy Sub

Prototype DECLARE SUB DQBpollJoy (BYVAL JoyNum)

Calling **Parameter** **Description**

JoyNum Joystick number to retrieve information of

Returns None.

Description This function must always be called before calling **DQBjoyMove** or **DQBjoyFire**, as it updates the internal joystick variables.

JoyNum **Description**

0 Joystick 1

1 Joystick 2

2 4-Button joystick or gamepad

There are three constants for JoyNum, as explained in Appendix A.

Notes Polling a joystick is a relatively slow operation. In addition, if the specified joystick is not connected, this will be even slower, so it is recommended that you check if the joystick is available before calling **DQBpollJoy**.

Example See **DQBjoyMove** example.

DQBpPut Sub

Prototype DECLARE SUB DQBpPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff, BYVAL Pattern)

Calling **Parameter** **Description**

Layer Layer to draw the sprite on

x x position of upper left corner of the sprite

y y position of upper left corner of the sprite

SpriteSeg Segment of the array holding the sprite data (use **VARSEG**)

SpriteOff Offset of the array holding the sprite data (use **VARPTR**)

Pattern Bit pattern (see below)

Returns None.

Description **DQBpPut** draws a sprite like **DQBput**, but uses the specified 8-bit pattern to determine which pixels are to be drawn. So if the binary pattern is 01010101 (85 in

decimal format), every other pixel will be drawn. The pattern wraps until the end of each sprite line, and on every new line it's XORed by 255. This means that the same example pattern will look like 10101010 on the second sprite line, and again as 01010101 on the third, etc.

Notes By alternating the bit-pattern and drawing sprites quickly using this function, you can achieve translucency-like effects, but without the need of a blender map. Use it wisely!

Example None.

DQBprint Sub

Prototype DECLARE SUB DQBprint (Layer AS INTEGER, Text AS STRING, x AS _
INTEGER, y AS INTEGER, Col AS INTEGER)

Calling **Parameter** **Description**

Layer	Layer to draw the text on
Text	String to print
x	x position of upper left corner of the first character
y	y position of upper left corner of the first character
Col	Text color

Returns None.

Description This function will print the specified string onto the given layer. By default, the VGA BIOS font is used, but any font can be used by calling **DQBsetFont** first. Also, the last text style selected by **DQBsetTextStyle** is used. The x and y parameters are in pixel units, so you can print the text anywhere on the specified layer. If you specify a x value of &H8000 (or you use the constant CENTERED - see Appendix A) the text is automatically centered on the current viewport set by the last call to **DQBsetClipBox** (which is, by default, the entire screen).

Notes **DQBprint** supports clipping, so pixels outside the clipping box will not be drawn. See also **DQBprints**, **DQBsetTextStyle**, **DQBsetFont**, and **DQBsetBIOSfont**.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE:'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Print text on the center of the screen in bright red
```

```

DQBprint VIDEO, "This is a DQBprint test!", CENTERED, 96, 40

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBprints Sub

Prototype DECLARE SUB DQBprints (Layer AS INTEGER, Text AS STRING, x AS _
INTEGER, y AS INTEGER, Col AS INTEGER, Style AS INTEGER)

Calling **Parameter** **Description**

Layer	Layer to draw the text on
Text	String to print
x	x position of upper left corner of the first character
y	y position of upper left corner of the first character
Col	Text color
Style	Custom text style

Returns None.

Description **DQBprints** works exactly like **DQBprint**, but it requires an additional parameter that specifies the style of the output text. The style is applied only to one call of this function. Once the text has been drawn, the original text style is restored. For the different styles, see Appendix A for a list of constants, and take a look at the **DQBsetTextStyle** function.

Notes **DQBprints** supports clipping, so pixels outside the clipping box will not be drawn. See also **DQBprint**, **DQBsetTextStyle**, **DQBsetFont**, and **DQBsetBIOSfont**.

Example

```

' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Use the normal font with DQBprint
DQBprint VIDEO, "This is a normal text", 0, 0, 31

' Use the normal font with the bold style
DQBprints VIDEO, "I'm big and fat, I'm bold style!", 0, 10, 31, BOLD
' Use italic style on the same font
DQBprints VIDEO, "Italic style is very nice", 0, 20, 31, ITALIC

```

```

' Use underlined style again on the same font
DQBprints VIDEO, "Pay attention to underlined text", 0, 30, 31, _
UNDERLINED
' Use a combination of styles on the same font
DQBprints VIDEO, "Bold and italic together!", 0, 40, 31, BOLD _
+ ITALIC

' Back to normal text
DQBprint VIDEO, "That's all folks!", 0, 50, 31

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBpset Sub

Prototype DECLARE SUB DQBpset (BYVAL Layer, BYVAL x, BYVAL y, BYVAL Col)

Calling	Parameter	Description
---------	-----------	-------------

Layer	Layer to draw the pixel on
x	Pixel x coordinate
y	Pixel y coordinate
Col	Pixel color

Returns None.

Description Sets a pixel on the specified layer, at the given coordinates and with Col color.

Notes This function is affected by the clipping box set by **DQBsetClipBox**.

Example None.

DQBput Sub

Prototype DECLARE SUB DQBput (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff)

Calling	Parameter	Description
	Layer	Layer to draw the sprite on
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)
Returns	None.	
Description	Draws the sprite contained in the given array on the specified layer, at the given coordinates. DQBput can operate both in transparent and solid mode. When the first mode is set, pixels with color 0 of the sprite are skipped, allowing a transparent put. When the second mode is set, all the pixels are copied to the layer, destroying the background. By default the put mode is set to transparent, but it can be changed at any time by simply calling DQBsetTransPut and DQBsetSolidPut.	
Notes	DQBput is affected by the clipping box, so pixels outside this box will not be drawn. The sprite data format is compatible with normal GET and PUT, so you can get your sprite data by calling either GET or DQBget. See also DQBfPut, DQBsPut, DQBrPut, and DQBbPut.	
Example	<pre>' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with one layer, no sounds, and no user EMS IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror\$: END ' We're going to use 15 sprite 16x16 pixels each; let's call DQBsize ' to know the dimension of our sprite array (we'll store all the ' sprites into the same array) in bytes Size = DQBsize(0, 0, 15, 15) ' Our array is made of INTEGERS, so we divide Size by 2 DIM Sprite((Size \ 2) * 15) ' Prepare our sprites FOR i = 0 TO 14 ' Clear layer 1 DQBclearLayer 1 ' Draw a frame of our sprite DQBbox 1, 0, 0, 15, 15, 4 DQBline 1, (15-i), 15, i, 0, 40 DQBline 1, 15, i, 0, (15 - i), 40 ' Save the frame into our array DQBget 1, 0, 0, 15, 15, VARSEG(Sprite(0)), VARPTR(Sprite(0)) _ + (Size * i) NEXT i</pre>	

```

DQBinitVGA

FOR i = -16 TO 320

    ' Clear layer 1
    DQBclearLayer 1

    ' Find what frame to draw
    Frame = ((i + 16) \ 2) MOD 15

    ' Draw our frame into layer 1
    DQBput 1, i, 92, VARSEG(Sprite(0)), (VARPTR(Sprite(0)) + (Size _
* Frame))

    ' Wait for vertical retrace 1 time
    DQBwait 1

    ' Copy layer 1 onto the screen
    DQBcopyLayer 1, VIDEO

NEXT i

' End the program
DQBclose
END

```

DQBputOver Sub

Prototype DECLARE SUB DQBputOver (BYVAL BackSeg, BYVAL BackOff, BYVAL x, _
BYVAL y, BYVAL SpriteSeg, BYVAL SpriteOff)

Calling	Parameter	Description
	BackSeg	Segment of the array holding the first sprite data (use VARSEG)
	BackOff	Offset of the array holding the first sprite data (use VARPTR)
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the second sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the second sprite data (use VARPTR)

Returns None.

Description This function draws a sprite over another one. The first sprite acts as the background, and the second sprite is drawn over it. **DQBputOver** supports transparency, and the second sprite is automatically clipped to the first one.

Notes None.

Example None.

DQBreadBit Function

Prototype DECLARE FUNCTION DQBreadBit (BYVAL Value, BYVAL Bit)

Calling	Parameter	Description
	Value	Integer value to read bit status from
	Bit	Bit number, ranging 0-15, to retrieve status of
Returns	True if specified bit is set, otherwise false.	
Description	This function returns true if the specified bit of the specified integer value is set. This can be useful to store Boolean variables in an integer variable, as QB does not support Booleans. An integer can contain up to 16 Boolean variables!	
Notes	See DQBsetBit , DQBresetBit , and DQBtoggleBit .	
Example	None.	

DQBreadKey Function

Prototype DECLARE FUNCTION DQBreadKey ()

Calling	None.
Returns	An integer value holding the scancode number of the key that has been pressed.
Description	DQBreadKey waits for the user to press a key, and then returns its scancode. This function works only if the keyboard interrupt handler has been installed by calling DQBinstallKeyboard .
Notes	See also DQBinstallKeyboard , DQBkey , and DQBwaitKey .
Example	None.

DQBremoveBMap Sub

Prototype DECLARE SUB DQBremoveBMap (BYVAL BMap)

Calling	Parameter	Description
	BMap	Blender map number to deallocate

Returns	None.
Description	Deallocates memory used by the specified blender map and turns it off.
Notes	This function is automatically called by DQBclose for every active blender map. See also DQBcreateBMap .
Example	None.

DQBremoveKeyboard Sub

Prototype	DECLARE SUB DQBremoveKeyboard ()
Calling	None.
Returns	None.
Description	Turns off the custom keyboard interrupt handler if it was on, giving the keyboard control back to the old BIOS routines. This function is automatically called by DQBclose .
Notes	See also DQBinstallKeyboard .
Example	See DQBkey example.

DQBremoveSB Sub

Prototype	DECLARE SUB DQBremoveSB ()
Calling	None.
Returns	None.
Description	Stops all the sounds playing and turns off the sound engine. This function is automatically called by DQBclose , so there's no particular need to call it.
Notes	See also DQBinstallSB .
Example	None.

DQBresetBit Function

Prototype DECLARE FUNCTION DQBresetBit (BYVAL Value, BYVAL Bit)

Calling	Parameter	Description
	Value Bit	Integer variable to reset bit of Bit number to reset, ranging 0-15
Returns	An integer value representing the variable value with specified bit reset.	
Description	Call this function to zero a bit of the specified variable. This can be useful when handling Boolean variables.	
Notes	See DQBsetBit , DQBreadBit , and DQBtoggleBit .	
Example	None.	

DQBresetJoy Sub

Prototype DECLARE SUB DQBresetJoy ()

Calling None.

Returns None.

Description Resets the joysticks by recalibrating them (if they are available).

Notes None.

Example None.

DQBresetMouse Sub

Prototype DECLARE SUB DQBresetMouse ()

Calling None.

Returns None.

Description Resets the default mouse cursor shape and sets the mouse range to (0, 0)-(319, 199). This function should be called just after entering VGA mode 13h.

Notes None.

Example None.

DQBresumeSound Sub

Prototype DECLARE SUB DQBresumeSound ()

Calling None.

Returns None.

Description Resume sound output paused by **DQBpauseSound**.

Notes See also **DQBpauseSound**.

Example None.

DQBrPut Sub

Prototype DECLARE SUB DQBrPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff, BYVAL Angle, BYVAL Zoom)

Calling	Parameter	Description
	Layer	Layer to draw the sprite on
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)
	Angle	Rotation angle ranging from 0-255
	Zoom	Zoom factor

Returns None.

Description **DQBrPut** draws a sprite rotated and scaled to the specified values. The angle can range from 0 to 255 (256 = 0), while the zoom factor represents the zooming percentage. A value of 100 will draw the sprite in its original size, a value of 50 will draw a sprite of half its original size, etc. The rotation uses a pixel-perfect algorithm, but rotated pixels that lie outside the original sprite mask will not be drawn. This means that if you want to rotate a sprite without losing graphics, you must get it into a larger area. For example, let's suppose you have a 16x16 pixel sprite. You should get an area of 22x22 pixels to be sure not to lose any pixel when rotating. The same goes when zooming in; the original sprite area is respected.

Notes	DQBrPut supports clipping and transparency. See also DQBput , DQBfPut , DQBsPut , and DQBbPut .
Example	<pre> ' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with one layer, no sounds, and no user EMS IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror\$: END ' Let's use an 80x80 pixel sprite Size = DQBsize(0, 0, 79, 79) ' Our array is made of INTEGERS, so we divide Size by 2 DIM Sprite(Size \ 2) ' Prepare our sprite DQBclearLayer 1 DQBprint 1, "DirectQB!", 8, 36, 40 DQBget 1, 0, 0, 79, 79, VARSEG(Sprite(0)), VARPTR(Sprite(0)) DQBinitVGA FOR i = 0 TO 90 ' Empty layer 1 DQBclearLayer 1 ' Draw our roto-zoomed sprite into layer 1 DQBrPut 1, 120, 60, VARSEG(Sprite(0)), VARPTR(Sprite(0)), (90 - _ i), (10 + i) ' Wait for vertical retrace 1 time DQBwait 1 ' Copy layer 1 onto the screen DQBcopyLayer 1, VIDEO NEXT i ' End the program DQBclose END </pre>

DQBsaveBMap Function

Prototype DECLARE FUNCTION DQBsaveBMap (BMap AS INTEGER, FileName AS STRING)

Calling	Parameter	Description
	BMap	Number of the blender map to save
	FileName	Blender map filename with full extension

Returns	Integer	Description
	0	Operation successful
	1	Blender map not yet created
	2	Cannot create file or disk error
	3	Unable to write data or disk full
Description	This function saves the specified blender map data to disk. The blender map can then be loaded again in memory by calling DQBlloadBMap .	
Notes	See also DQBcreateBMap and DQBlloadBMap .	
Example	None.	

DQBsavelImage Function

Prototype DECLARE FUNCTION DQBsavelImage (Layer AS INTEGER, x1 AS INTEGER, y1 _
 AS INTEGER, x2 AS INTEGER, y2 AS INTEGER, FileName AS STRING, Pal _
 AS STRING, Format AS INTEGER)

Calling	Parameter	Description
	Layer	Layer the image to save is on
	x1	x coordinate of upper-left area of layer to save
	y1	y coordinate of upper-left area of layer to save
	x2	x coordinate of lower-right area of layer to save
	y2	y coordinate of lower-right area of layer to save
	FileName	New image filename with full extension
	Pal	String holding the palette to save
	Format	Image file format

Returns	Integer	Description
	0	Operation successful
	1	Cannot create file or disk error
	2	Unable to write data or disk full

Description Call this function to save an area of a layer to a file. Images can be saved in three different formats: **BSAVE** (BSV) compatible, bitmap (BMP), or PC Paintbrush (PCX). When saving in the **BSAVE** format, you can load the image by simply calling the QB command **BLOAD**, and the palette will be appended to the end of the file. In addition, **BSAVE** images are large files, but they load the fastest. When using the BMP format, the resulting file will be even larger, but you will have a image saved in a standard file format, accessible by most commercial image-editing programs. PCX is the smallest available file format, and it also contains the palette, but it also loads the slowest. The palette to save with the file must have been previously stored in a

string as explained in Appendix A, by calling **DQBgetPal** or even **DQBloadImage**. This is done to allow saving hidden layers with a palette different from the current one.

Notes When you're saving a 320x200 image in the **BSAVE** format, it will be saved just like the QB command:

```
DEF SEG = layerseg
BSAVE imagefile, 0, 64000
```

So you'll be able to load it on the screen without requiring an intermediate buffer. On the other hand, if you use the **BSAVE** format to save an image that is smaller than 320x200, it will be like if you used:

```
GET (x1, y1) - (x2, y2), array
DEF SEG = VARSEG(array)
BSAVE imagefile, VARPTR(array), length
```

In both cases, the palette will be appended to the file. Don't worry, however, because **BLOAD** will be still able to load your images—though without loading the appended palette.

Example None.

DQBscroll Sub

Prototype DECLARE SUB DQBscroll (BYVAL Layer, BYVAL dx, BYVAL dy)

Calling	Parameter	Description
	Layer	Layer to be scrolled
	dx	Horizontal scrolling offset
	dy	Vertical scrolling offset

Returns None.

Description Scrolls the selected layer. A positive dx value will scroll the layer right, a negative value will scroll it left. A positive dy value scrolls down; a negative value scrolls up. By combining these parameters, you can scroll the layer in any direction. The newly visible area will be filled with garbage; it's up to the programmer to fill it with new graphics.

Notes None.

Example None.

DQBscrollArea Sub

Prototype DECLARE SUB DQBscrollArea (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL _
x2, BYVAL y2, BYVAL Direction)

Calling	Parameter	Description
---------	-----------	-------------

Layer	Layer to be scrolled
x1	x coordinate of upper-left corner of area to scroll
y1	y coordinate of upper-left corner of area to scroll
x2	x coordinate of lower-right corner of area to scroll
y2	y coordinate of lower-right corner of area to scroll
Direction	Scrolling direction

Returns None.

Description Scrolls the specified area on the given layer. The area can only be scrolled one pixel at a time. The direction parameter can range from 0 to 3, representing up, down, left, and right scrolling. You can use the UP, DOWN, LEFT, and RIGHT constants defined in the DIRECTQB.BI file for ease.

Notes None.

Example None.

DQBsetBaseLayer Function

Prototype DECLARE FUNCTION DQBsetBaseLayer (BYVAL Layer)

Calling	Parameter	Description
---------	-----------	-------------

Layer	Layer in base memory to allocate
-------	----------------------------------

Returns 0 if the base layer was already allocated or there's not enough free base memory, otherwise the segment of the new memory block used as the layer.

Description **DQBsetBaseLayer** tries to allocate base memory for a new base layer, as specified by the Layer parameter. If successful, this function returns a segment address to the allocated memory chunk used as layer, which is exactly 64,000 bytes. You can then refer to this layer as you do with normal layers in EMS memory.

Notes Keep in mind that base layers require a lot of base memory, but they are accessed a lot faster than normal EMS layers. Also, before calling **DQBsetBaseLayer**, you should free 64K of base memory by calling the QB function **SETMEM**. To refer to base layers you should always use the B0...B9 constants as explained in Appendix A.

Example	<pre> ' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with no layers, no sounds, and no user EMS IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror\$: END ' Release memory for a base layer AllocMem& = SETMEM(-65537) ' Assign our array to base layer 0 Seg0 = DQBsetBaseLayer(B0) IF Seg0 = 0 THEN ' Not enough memory DQBclose PRINT DQBerror\$ END ELSE ' Print some info PRINT "Layer allocated at address " + HEX\$(Seg0) + ":0000" WHILE INKEY\$ = "": WEND END IF DQBinitVGA ' Print some text on layer B0 DQBprint B0, "DirectQB!", 124, 96, 40 ' Copy layer B0 to the screen DQBcopyLayer B0, VIDEO ' Wait for the user to press a key... WHILE INKEY\$ = "": WEND ' End the program DQBclose FreeMem& = SETMEM(65537) END </pre>
----------------	--

DQBsetBIOSfont Sub

Prototype	DECLARE SUB DQBsetBIOSfont ()
Calling	None.
Returns	None.
Description	Resets the current font to the standard VGA BIOS font. Future calls to DQBprint will use the BIOS font.
Notes	None.
Example	See DQBprint example.

DQBsetBit Function

Prototype DECLARE FUNCTION DQBsetBit (BYVAL Value, BYVAL Bit)

Calling	Parameter	Description
	Value	Integer variable to set bit of
	Bit	Bit number to set, ranging 0-15
Returns	An integer value representing the variable value with the new bit set.	
Description	Call this function to set a bit of a specified variable. This can be useful when handling Boolean variables.	
Notes	See DQBresetBit , DQBreadBit , and DQBtoggleBit .	
Example	None.	

DQBsetBMap Sub

Prototype DECLARE SUB DQBsetBMap (BYVAL BMap, BYVAL ForeCol, BYVAL BackCol, _
BYVAL NewCol)

Calling	Parameter	Description
	BMap	Blender map number to edit
	ForeCol	Foreground color
	BackCol	Background color
	NewCol	Color of the resulting combination
Returns	None.	
Description	Sets a color for the specified combination on the specified blender map. Never try to set the combinations for an unmapped foreground color!	
Notes	See also DQBcreateBMap , DQBgetBMap , DQBloadBMap , and DQBsaveBMap .	
Example	See DQBbPut example.	

DQBsetClipBox Sub

Prototype DECLARE SUB DQBsetClipBox (BYVAL x1, BYVAL y1, BYVAL x2, BYVAL y2)

Calling	Parameter	Description
	x1	Upper left corner x coordinate of clipping box
	y1	Upper left corner y coordinate of clipping box
	x2	Lower right corner x coordinate of clipping box
	y2	Lower right corner y coordinate of clipping box
Returns	None.	
Description	Sets the current clipping box to (x1, y1)-(x2, y2). Future calls to all the drawing functions (except DQBbox and DQBboxf) will be affected by the new clipping box.	
Notes	The clipping box is set to (0, 0)-(319, 199) at startup.	
Example	None.	

DQBsetCol Sub

Prototype DECLARE SUB DQBsetCol (BYVAL Col, BYVAL Red, BYVAL Green, BYVAL _ Blue)

Calling	Parameter	Description
	Col	Color to change
	Red	New red hue
	Green	New green hue
	Blue	New blue hue
Returns	None.	
Description	Sets the hues of a given color index. The red, green, and blue hues must be in the range 0-63.	
Notes	None.	
Example	None.	

DQBsetCollideMethod Sub

Prototype DECLARE SUB DQBsetCollideMethod (BYVAL Method)

Calling	Parameter	Description
	Method	Collision detection method to be used

Returns	None.
Description	Sets the collision detection method to be used by the DQBcollide function. The method parameter can be 0 (bounding box check) or 1 (pixel-perfect check). You can use the BOX and PIXEL constants for ease.
Notes	None.
Example	None.

DQBsetDrive Sub

Prototype DECLARE SUB DQBsetDrive (Drive AS STRING)

Calling	Parameter	Description
	Drive	New drive letter

Returns None.

Description This function changes the current drive to the new one specified by the letter of the Drive parameter.

Notes See also **DQBdrive\$**, **DQBchDir**, **DQBdir\$**, **DQBpath\$**, and **DQBnumDrives**.

Example None.

DQBsetFont Sub

Prototype DECLARE SUB DQBsetFont (Font AS STRING)

Calling	Parameter	Description
	Font	String of 2305 characters holding the new font data

Returns None.

Description Sets the current font to specified one held in the Font string (see Appendix C for details on the font data format). Future calls to **DQBprint** or **DQBprints** will use the new font to draw characters.

Notes You can use DirectQB Tools to create your own fonts. See also **DQBsetBIOSfont**.

Example None.

DQBsetFontTexture Sub

Prototype DECLARE SUB DQBsetFontTexture (BYVAL TextSeg, BYVAL TextOff)

Calling	Parameter	Description
	TextSeg	Segment of the array holding the texture sprite (use VARSEG)
	TextOff	Offset of the array holding the texture sprite (use VARPTR)

Returns None.

Description Sets the texture to be used whenever the TEXTURED font style is used. Font textures must be normal sprites stored with **GET** or **DQBget**, but their size must always be 8x8 pixels, otherwise you may end up with unpredictable results. The specified texture is then used to fill the active pixels of each character of your string.

Notes None.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Dimension our 8x8 pixel texture array
DIM Texture(35)

' Initialize the library with one layer, no sounds, and no user EMS
IF DQBinit(1, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Prepare the sprite that will be used as font texture on layer 1
FOR i = 0 TO 7
    DQBline 1, 0, i, i, 0, (31 - i)
NEXT i
FOR i = 0 TO 7
    DQBline 1, 7, i, i, 7, (24 - i)
NEXT i

' Get the sprite!
DQBget 1, 0, 0, 7, 7, VARSEG(Texture(0)), VARPTR(Texture(0))

' Let's set the font texture to our new one
DQBsetFontTexture VARSEG(Texture(0)), VARPTR(Texture(0))

' Print a string on the screen
DQBprints VIDEO, "This is a textured font test!!", 0, 100, 15,
TEXTURED

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
```

END

DQBsetFrameRate Sub

Prototype DECLARE SUB DQBsetFrameRate (BYVAL FPS)

Calling	Parameter	Description
	FPS	New frames per second selection

Returns None.

Description Sets a new frame rate. Together with the **DQBframeReady** function, this function can be used to synchronize the graphics speed to a specified frame rate, in an independent way from the program complexity.

Notes **DQBsetFrameRate** alters the PC timer speed, so use caution when calling this function; it may not be compatible with some TSRs, like SBMIDI. See also **DQBframeReady**.

Example None.

DQBsetJoySens Sub

Prototype DECLARE SUB DQBsetJoySens (BYVAL NewSens)

Calling	Parameter	Description
	NewSens	New joystick sensitivity setting

Returns None.

Description Changes the joystick sensitivity.

Notes None.

Example None.

DQBsetMousePos Sub

Prototype DECLARE SUB DQBsetMousePos (BYVAL x, BYVAL y)

Calling	Parameter	Description
	x	x coordinate of new mouse position
	y	y coordinate of new mouse position

Returns None.

Description Sets the new mouse position to (x, y).

Notes None.

Example None.

DQBsetMouseRange Sub

Prototype DECLARE SUB DQBsetMouseRange (BYVAL x1, BYVAL y1, BYVAL x2, BYVAL y2)

Calling	Parameter	Description
	x1	Upper left corner x coordinate of new range box
	y1	Upper left corner y coordinate of new range box
	x2	Lower right corner x coordinate of new range box
	y2	Lower right corner y coordinate of new range box

Returns None.

Description Sets the new mouse range to (x1, y1)-(x2,y2).

Notes The mouse range is set to (0, 0)-(319, 199) at startup.

Example None.

DQBsetMouseShape Sub

Prototype DECLARE SUB DQBsetMouseShape (hotX AS INTEGER, hotY AS INTEGER, _
Shape AS STRING)

Calling	Parameter	Description
	hotX	New cursor x hotspot coordinate
	hotY	New cursor y hotspot coordinate
	Shape	New cursor shape data string
Returns	None.	
Description	Sets the new cursor shape to the specified one. There's no need to hide the mouse before calling this function, as this is automatically done for you by this function. See Appendix C for details on the mouse cursor shape data format.	
Notes	Call DQBresetMouse to restore the default mouse cursor shape. This will also reset the mouse range box to (0, 0)-(319, 199), so be warned.	
Example	None.	

DQBsetMouseSpeed Sub

Prototype DECLARE SUB DQBsetMouseSpeed (BYVAL Hor, BYVAL Ver)

Calling	Parameter	Description
	Hor	New horizontal mouse speed
	Ver	New vertical mouse speed
Returns	None.	
Description	Sets the new mouse horizontal and vertical speed. By default, both speeds are set to 16. Smaller values will make the mouse movement faster; larger values will make the mouse movement slower.	
Notes	None.	
Example	None.	

DQBsetPal Sub

Prototype DECLARE SUB DQBsetPal (Pal AS STRING)

Calling	Parameter	Description
	Pal	New palette data string

Returns	None.
Description	Sets the current palette to the new specified one. See Appendix C for details on the palette data format.
Notes	See also DQBgetPal .
Example	See DQBfadeTo example.

DQBsetSolidPut Sub

Prototype	DECLARE SUB DQBsetSolidPut ()
Calling	None.
Returns	None.
Description	Sets the solid put mode. Future calls to DQBput or any other DirectQB “put” routine will ignore transparent color.
Notes	See also DQBsetTransPut .
Example	See DQBput example.

DQBsetTextBackCol Sub

Prototype	DECLARE SUB DQBsetTextBackCol (BYVAL Col)	
Calling	Parameter	Description
	Col	New text background color
Returns	None.	
Description	Sets the current text background color. This will be used only when displaying solid text.	
Notes	By default, the text background color is 0. See also DQBsetTextStyle and DQBprint .	
Example	None.	

DQBsetTextBMap Sub

Prototype DECLARE SUB DQBsetTextBMap (BYVAL BMap)

Calling	Parameter	Description
---------	-----------	-------------

	BMap	Blender map to be used
--	------	------------------------

Returns None.

Description Sets the blender map to be used when the BLENDED text style is selected to print text with **DQBprint** or **DQBprints**.

Notes None.

Example None.

DQBsetTextSpacing Sub

Prototype DECLARE SUB DQBsetTextSpacing (BYVAL Spacing)

Calling	Parameter	Description
---------	-----------	-------------

	Spacing	Extra spacing between characters in pixels
--	---------	--

Returns None.

Description You can call this function to modify the spacing between characters drawn in the current font with **DQBprint** or **DQBprints**. By default this value is 0 (no extra spacing). By increasing it, you can add extra spacing between each character printed.

Notes None.

Example None.

DQBsetTextStyle Sub

Prototype DECLARE SUB DQBsetTextStyle (BYVAL Style)

Calling	Parameter	Description
---------	-----------	-------------

	Style	New text style
--	-------	----------------

Returns	None.
Description	Sets the current text style used by DQBprint . The style is independent from the current font. The available styles are bold, italic, and underlined. To use no styles just pass the NONE constant to this function. DQBsetTextStyle also allows you to set the transparent or solid text mode as well as support for color blending and textured text. By default, the transparent mode is set, but by passing the SOLID constant, future calls to DQBprint will print solid text. To obtain blended text, just add the BLENDED constant, while adding TEXTURED causes the current font texture to be used to print your text. You can use combinations of all the styles, by simply adding the desired constants, found in Appendix A.
Notes	See also DQBprint , DQBprints , and DQBsetFontTexture .
Example	<pre>' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with no layers, no sounds, and no user EMS IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror\$: END DQBinitVGA ' Use the normal font with DQBprint DQBprint VIDEO, "This is a normal text", 0, 0, 31 ' Let's set solid italic style... DQBsetTextStyle SOLID + ITALIC ' ...and red background color DQBsetTextBackCol 4 ' Print something using the new style DQBprint VIDEO, "And this is solid italic text!", 0, 50, 31 ' Wait for the user to press a key WHILE INKEY\$ = "": WEND ' End the program DQBclose END</pre>

DQBsetTextureSize Sub

Prototype	DECLARE SUB DQBsetTextureSize (BYVAL Size)	
Calling	Parameter	Description
Returns	Size	New texture width
	None.	

Description	Contrary to its name, DQBsetTextureSize is used to set only the width of the texture, used by subsequent calls to DQBttri . The width must be specified in pixels and must be a power of 2; values that are not powers of 2 will be rounded to the previous power. So if you specify 63, the function will use 32, and if you specify 64, it will use 64, etc. There are no limits concerning the height of a texture, you only have to worry about the texel coordinates of the vertices of your triangle.
Notes	By default, the texture size is set to 64. See also DQBttri .
Example	None.

DQBsetTransPut Sub

Prototype	DECLARE SUB DQBsetTransPut ()
Calling	None.
Returns	None.
Description	Sets the transparent put mode. Future calls to DQBput or any other DirectQB “put” routine will draw transparent sprites.
Notes	See also DQBsetSolidPut .
Example	See the DQBput example.

DQBsetVoiceVol Sub

Prototype	DECLARE SUB DQBsetVoiceVol (BYVAL Voice, BYVAL Volume)	
Calling	Parameter	Description
	Voice Volume	Voice to change volume of New voice volume setting
Returns	None.	
Description	Sets the volume setting for a specified voice. The voice volume may range from 0 to 63.	
Notes	By default, the volume setting for all the voices is 63. Remember that the volume of a voice also depends on the master volume setting. This function has no effect if the volume table is off (see DQBinstallSB for details).	

Example None.

DQBsetVolume Sub

Prototype DECLARE SUB DQBsetVolume (BYVAL Volume)

Calling	Parameter	Description
---------	-----------	-------------

	Volume	New master volume setting
--	--------	---------------------------

Returns None.

Description Sets the master volume setting for sound output. The master volume may range from 0 to 15.

Notes There is no default volume setting, so it is recommended that you set the volume as you wish just after calling **DQBinstallSB** in your programs.

Example None.

DQBshiftLeft Function

Prototype DECLARE FUNCTION DQBshiftLeft (BYVAL Value, BYVAL NumBits)

Calling	Parameter	Description
---------	-----------	-------------

	Value	Integer value to shift
	NumBits	Number of bits to shift left

Returns An Integer value representing the shifted number

Description Performs a logical shift left of the given integer value, and returns the result. This can be useful to multiply by powers of 2, as this operation is faster than *.

Notes See also **DQBshiftRight**.

Example None.

DQBshiftRight Function

Prototype DECLARE FUNCTION DQBshiftRight (BYVAL Value, BYVAL NumBits)

Calling	Parameter	Description
	Value	Integer value to shift
	NumBits	Number of bits to shift right
Returns	An integer value representing the shifted number	
Description	Performs a logical shift right of the given integer value, and returns the result. This can be useful to divide by powers of 2, as this operation is faster than \.	
Notes	See also DQBshiftLeft .	
Example	None.	

DQBsize Function

Prototype DECLARE FUNCTION DQBsize (BYVAL x1, BYVAL y1, BYVAL x2, BYVAL y2)

Calling	Parameter	Description
	x1	Upper left corner x coordinate
	y1	Upper left corner y coordinate
	x2	Lower right corner x coordinate
	y2	Lower right corner y coordinate
Returns	An Integer value holding the sprite size in bytes	
Description	DQBsize returns the number of bytes needed to store the image contained in the area (x1, y1)-(x2, y2) with DQBget . This value can be then used to dimension arrays to store the sprite data for the DQBget and DQBput routines. Remember that the array is made of integers, while this function returns the number of bytes. To obtain the array dimension, divide the result by 2 and subtract 1.	
Notes	This function can be useful when storing several sprites in the same array, allowing the programmer to know the exact offset of each frame.	
Example	See DQBput example.	

DQBsort Sub

Prototype DECLARE SUB DQBsort (BYVAL ArraySeg, BYVAL ArrayOff, BYVAL _
NumRecords, BYVAL RecordLen, IndexOff)

Calling	Parameter	Description
	ArraySeg	Segment of the array of integers to sort (use VARSEG)
	ArrayOff	Offset of the array of integers to sort (use VARPTR)
	NumRecords	Total number of records to sort
	RecordLen	Length of each record in bytes
	IndexOff	Offset of the integer index value within a record

Returns None.

Description **DQBsort** uses a fast bubble sort algorithm to sort the given array of records. The records are sorted by comparing the index number (must be an integer) of each record and sorting the corresponding records from the lowest to highest. The index field can be anywhere within the record; you can specify its position by using the IndexOff parameter. This is relative to the beginning of a record (i.e. IndexOff = 0 means the first field of the record). RecordLen is the number of bytes of each record.

Notes A record can contain any type of data; **DQBsort** will sort it by comparing the index numbers. You can also sort a simple array of integers, by passing 2 as RecordLen (an integer is 2 bytes long) and 0 as IndexOff.

Example None.

DQBSPut Sub

Prototype DECLARE SUB DQBSPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff, BYVAL NewWidth, BYVAL NewHeight)

Calling	Parameter	Description
	Layer	Layer to draw the sprite on
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)
	NewWidth	New sprite width in pixel units
	NewHeight	New sprite height in pixel units

Returns None.

Description **DQBsPut** works like **DQBput**, but draws a scaled version of your sprite by specifying a new width and height. The sprite data is not modified during this operation. This function also supports clipping and transparency.

Notes See **DQBput**, **DQBfPut**, **DQBrPut**, and **DQBbPut**.

Example See **DQBput** example.

DQBstopVoice Sub

Prototype DECLARE SUB DQBstopVoice (BYVAL Voice)

Calling	Parameter	Description
---------	-----------	-------------

	Voice	Voice to stop
--	-------	---------------

Returns None.

Description Stops any sound coming from specified voice. You can use this to stop a looping sample.

Notes None.

Example None.

DQBtoggleBit Function

Prototype DECLARE FUNCTION DQBtoggleBit (BYVAL Value, BYVAL Bit)

Calling	Parameter	Description
---------	-----------	-------------

	Value	Integer value to toggle bit of
	Bit	Bit number to toggle

Returns An integer value representing the original number with specified bit toggled.

Description Toggles specified bit of the given number, and returns the result.

Notes See also **DQBsetBit**, **DQBresetBit**, and **DQBreadBit**.

Example None.

DQBtPut Sub

Prototype DECLARE SUB DQBtPut (BYVAL Layer, BYVAL x, BYVAL y, BYVAL _
SpriteSeg, BYVAL SpriteOff, BYVAL BitMode)

Calling	Parameter	Description
	Layer	Layer to draw the sprite on
	x	x position of upper left corner of the sprite
	y	y position of upper left corner of the sprite
	SpriteSeg	Segment of the array holding the sprite data (use VARSEG)
	SpriteOff	Offset of the array holding the sprite data (use VARPTR)
	BitMode	Specifies which bitwise operation to perform

Returns None.

Description Like **DQBput**, this routine draws a sprite on the specified layer. However, **DQBtPut** allows the programmer to apply a specified bitwise operation between each pixel of your sprite and the background. Supported modes are AND, OR, and XOR, represented by the numbers 1, 2, and 3 (you can use the BIT.AND, BIT.OR, and BIT.XOR constants here—see Appendix A). This function supports both transparency and clipping.

Notes None.

Example None.

DQBtri Sub

Prototype DECLARE SUB DQBtri (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, _
BYVAL y2, BYVAL x3, BYVAL y3, BYVAL Col)

Calling	Parameter	Description
	Layer	Layer to draw the triangle on
	x1	x coordinate of the first vertex
	y1	y coordinate of the first vertex
	x2	x coordinate of the second vertex
	y2	y coordinate of the second vertex
	x3	x coordinate of the third vertex
	y3	y coordinate of the third vertex
	Col	color

Returns None.

Description Draws a flat-shaded triangle with (x1, y1), (x2, y2), and (x3, y3) as vertices, filled with the specified color. **DQBtri** does not support transparency, but it is affected by the clipping box.

Notes See also **DQBgtri**.

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA

' Draw 500 flat-shaded triangles
FOR i = 1 TO 500
    DQBtri VIDEO, (RND * 320), (RND * 200), (RND * 320), (RND * _
200), (RND * 320), (RND * 200), (RND * 256)
NEXT i

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END
```

DQBttri Sub

Prototype DECLARE SUB DQBttri (BYVAL Layer, BYVAL x1, BYVAL y1, BYVAL x2, BYVAL y2, BYVAL x3, BYVAL y3, BYVAL u1, BYVAL v1, BYVAL u2, BYVAL v2, BYVAL u3, BYVAL v3, BYVAL TextureSeg, BYVAL TextureOff)

Calling	Parameter	Description
	Layer	Layer to draw the triangle on
	x1	x coordinate of the first vertex
	y1	y coordinate of the first vertex
	x2	x coordinate of the second vertex
	y2	y coordinate of the second vertex
	x3	x coordinate of the third vertex
	y3	y coordinate of the third vertex
	u1	x coordinate of first vertex on texture
	v1	y coordinate of first vertex on texture
	u2	x coordinate of second vertex on texture
	v2	y coordinate of second vertex on texture
	u3	x coordinate of third vertex on texture
	v3	y coordinate of third vertex on texture
	TextureSeg	Array segment holding the texture (use VARSEG)
	TextureOff	Array offset holding the texture (use VARPTR)

Returns None.

Description Draws a texture-mapped triangle with (x1, y1), (x2, y2), and (x3, y3) as vertices. The texture used must be stored in the specified array with **GET** or **DQBget**. The texture's width must be a power of 2, equal to the one specified by calling **DQBsetTextureSize** (the default texture width is 64 pixels). Each vertex of the triangle has also its own texture coordinates, which can be anywhere on the texture itself, as well as out of it. If the distance between two texture vertices is greater than the texture size, the texture is wrapped while drawing. **DQBttri** supports transparency (color 0 on the texture is always the transparent color) and clipping.

Notes See also **DQBsetTextureSize**, **DQBtri**, **DQBbtri**, and **DQBgtri**

Example

```
' All integers for speed
DEFINT A-Z

'$INCLUDE: 'DIRECTQB.BI'

' Dimension our 64x64 pixel texture array
DIM Texture(2049)

' Initialize the library with no layers, no sounds, and no user EMS
IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror$: END

DQBinitVGA
```

```

' Let's build our sample texture
FOR i = 0 TO 15
    DQBbox VIDEO, i, i, 63 - i, 63 - i, 31 - i
    DQBbox VIDEO, 16 + i, 16 + i, 47 - i, 47 - i, 16 + i
NEXT i
DQBbox VIDEO, 0, 0, 63, 63, 40
DQBline VIDEO, 0, 0, 63, 63, 4
DQBellipse VIDEO, 32, 32, 24, 24, 43

' Get our texture and clear the screen
DQBget VIDEO, 0, 0, 63, 63, VARSEG(Texture(0)), VARPTR(Texture(0))
DQBclearLayer VIDEO

' Draw 500 textured triangles
FOR i = 1 TO 500
    DQBttri VIDEO, (RND * 320), (RND * 200), (RND * 320), (RND * _
200), (RND * 320), (RND * 200), 0, 0, 63, 0, 0, 63, _
VARSEG(Texture(0)), VARPTR(Texture(0))
NEXT i

' Wait for the user to press a key
WHILE INKEY$ = "": WEND

' End the program
DQBclose
END

```

DQBunpackBMap Function

Prototype DECLARE FUNCTION DQBunpackBMap (BYVAL PacketID, BYVAL BMap)

Calling	Parameter	Description
	PacketID	Packet ID number
	BMap	Blender map to load data into
Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data
	4	Blender map is not active
	5	Incompatible blender map
Description	Decodes the blender map identified by the specified packet ID number from the currently open datafile. The blender map should be compatible with the map structure that you are going to decode it to (see DQBloadBMap for details).	
Notes	None.	

Example See **DQBopenDataFile**.

DQBunpackCursor Function

Prototype DECLARE FUNCTION DQBunpackCursor (PacketID AS INTEGER, Cursor AS _
STRING)

Calling	Parameter	Description
	PacketID	Packet ID number
	Cursor	64 character string to hold decoded cursor data

Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data

Description Decodes the mouse cursor identified by the specified packet ID number from the currently open datafile, and stores it in the Cursor string, ready to be used with the **DQBsetMouseShape** function.

Notes None.

Example See **DQBopenDataFile** example.

DQBunpackFont Function

Prototype DECLARE FUNCTION DQBunpackFont (PacketID AS INTEGER, Font AS _
STRING)

Calling	Parameter	Description
	PacketID	Packet ID number
	Font	2305 character string to hold decoded font data

Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data

Description Decodes the font identified by the specified packet ID number from the currently open datafile, and stores it in the Font string, ready to be used with the **DQBsetFont** function.

Notes None.

Example See **DQBopenDataFile** example.

DQBunpackImage Function

Prototype DECLARE FUNCTION DQBunpackImage (BYVAL PacketID, BYVAL Layer, _
BYVAL x, BYVAL y)

Calling	Parameter	Description
	PacketID	Packet ID number
	Layer	Layer to draw the image on
	x	x coordinate where to begin drawing
	y	y coordinate where to begin drawing

Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data

Description Decodes the image identified by the specified packet ID number from the currently open datafile, and draws it on the given layer, starting at (x, y). The image packet does not include a palette, so you will have to use a palette taken from another source (a palette packet from the same datafile, or any other way you like). Also, this function does not support clipping, so the programmer must ensure that the image fits on the layer.

Notes None.

Example See **DQBopenDataFile** example.

DQBunpackPal Function

Prototype DECLARE FUNCTION DQBunpackPal (PacketID AS INTEGER, Pal AS STRING)

Calling	Parameter	Description
	PacketID	Packet ID number
	Pal	768 character string to hold decoded palette data
Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data
Description	Decodes the palette identified by the specified packet ID number from the currently open datafile, and stores it in the Pal string. You can then use it with all the other DirectQB palette handling functions.	
Notes	None.	
Example	See DQBopenDataFile example.	

DQBunpackSound Function

Prototype DECLARE FUNCTION DQBunpackSound (BYVAL PacketID, BYVAL Slot)

Calling	Parameter	Description
	PacketID	Packet ID number
	Slot	Sound slot to place decoded sound data
Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data
	4	Unknown sound slot
Description	Decodes the sound sample identified by the specified packet ID number from the currently open datafile, and stores it in the specified sound slot. The slot must be initialized beforehand with DQBinit . Decoded sounds can then be normally played with DQBplaySound .	
Notes	None.	
Example	See DQBopenDataFile example.	

DQBunpackSprite Function

Prototype DECLARE FUNCTION DQBunpackSprite (BYVAL PacketID, BYVAL _
SpriteSeg, BYVAL SpriteOff)

Calling	Parameter	Description
	PacketID	Packet ID number
	SpriteSeg	Segment of QB array to hold decoded sprite data
	SpriteOff	Offset of QB array to hold decoded sprite data
Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data
Description This function is similar to DQBunpackImage , but it stores decoded image data in the specified QB array, ready to be used with DQBput or similar functions. The array must be large enough to hold the sprite, otherwise the computer may crash.		
Notes This function works with image packets. This means that you can call DQBunpackImage as well as DQBunpackSprite on the same image packet.		
Example See DQBopenDataFile example.		

DQBunpackUser Function

Prototype DECLARE FUNCTION DQBunpackUser (BYVAL PacketID, BYVAL DataSeg, _
BYVAL DataOff)

Calling	Parameter	Description
	PacketID	Packet ID number
	DataSeg	Segment of QB array to hold decoded user-defined data
	DataOff	Offset of QB array to hold decoded user-defined data
Returns	Integer	Description
	0	Operation successful
	1	Datafile not yet opened
	2	Unknown packet ID number
	3	Bad password or unknown data

Description	Decodes a user-defined data packet identified by the specified packet ID number from the currently open datafile in a QB array. User-defined packets can hold any kind of binary data. The maximum data size is 64K, however. The array to place decoded data in must be large enough to hold the data, or the computer may crash.
Notes	None.
Example	See DQBopenDataFile example.

DQBver Function

Prototype	DECLARE FUNCTION DQBver ()
Calling	None.
Returns	An integer value holding the library version.
Description	DQBver returns the current DirectQB library version in an integer value. The higher byte represents the major version, and the lower byte represents the minor version number.
Notes	DQBid\$ can also be used to retrieve a small string holding the library version and my name, in case you want to display them.
Example	<pre>' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with no layers, no sounds, and no user EMS IF DQBinit(0, 0, 0) THEN DQBclose: PRINT DQBerror\$: END ' Find major and minor library version numbers major = DQBver \ &H100 minor = DQBver AND &HFF ' Print them PRINT "Current DirectQB version is" + STR\$(major) + "." _ + LTRIM\$(STR\$(minor)) ' Print the library identification string PRINT DQBid\$ ' Wait for the user to press a key WHILE INKEY\$ = "": WEND ' End the program DQBclose END</pre>

DQBwait Sub

Prototype DECLARE SUB DQBwait (BYVAL Times)

Calling	Parameter	Description
	Times	Number of times to wait for the video vertical retrace

Returns None.

Description Waits for the video vertical retrace Times times. It is suggested to wait for the vertical retrace one time before drawing anything on the screen; this will help produce flickerless animations. The vertical retrace occurs 60 times per second, so you can also use this function as a delay routine.

Notes None.

Example None.

DQBwaitKey Sub

Prototype DECLARE SUB DQBwaitKey (BYVAL ScanCode)

Calling	Parameter	Description
	Scancode	Scancode of the key to wait for

Returns None.

Description Waits for the user to press the key specified by the scancode passed to the function. **DQBwaitKey** works only if the keyboard interrupt handler has been installed by calling **DQBinstallKeyboard**.

Notes See **DQBinstallKeyboard**, **DQBkey**, and **DQBreadKey**.

Example None.

DQBxPut Sub

Prototype DECLARE SUB DQBxPut (BYVAL SourceLayer, BYVAL x1, BYVAL y1, BYVAL _
x2, BYVAL y2, BYVAL DestLayer, BYVAL x, BYVAL y)

Calling	Parameter	Description
	SourceLayer	Layer to get the sprite from (source)
	x1	x position of upper left corner of the sprite on source
	y1	y position of upper left corner of the sprite on source
	x2	x position of lower right corner of the sprite on source
	y2	y position of lower right corner of the sprite on source
	DestLayer	Layer to draw the sprite on (destination)
	x	x position of upper left corner of the sprite on destination
	y	y position of upper left corner of the sprite on destination
Returns	None.	
Description	Like normal DQBput, this function draws a sprite on the specified layer, but with a huge difference. DQBxPut does not require that you first get your sprite in an array! DQBxPut automatically gets the data by assuming you want to draw the sprite contained in the area (x1 ,y1)-(x2, y2) on the source layer. Sprite data is directly copied to the destination layer at the specified (x, y) coordinates. This function is great, but it also has some weak spots. When both the source and destination layers are in EMS, DQBxPut is slower than DQBput. In addition, DQBxPut can only handle sprites with a height up to 50 pixels; higher sprites will not be drawn (but there are no limits for the width).	
Notes	DQBxPut supports transparency and is affected by the clipping box, so pixels outside this box will not be drawn. It is strongly suggested that you use this function to draw from an EMS layer onto a layer in base memory. This way, you can achieve almost the same speed of DQBput, without the need of extra memory to hold the sprite. See also DQBput, DQBfPut, DQBsPut, DQBrPut, and DQBbPut.	
Example	<pre>' Here's a modified version of the DQBput Example; this time we'll ' store our sprites without using arrays... ' All integers for speed DEFINT A-Z '\$INCLUDE: 'DIRECTQB.BI' ' Initialize the library with two layers, no sounds, and no user EMS IF DQBinit(2, 0, 0) THEN DQBclose: PRINT DQBerror\$: END ' Let's prepare our sprites on layer 2 FOR i = 0 TO 14 ' Draw a frame of our sprite DQBbox 2, (i * 16), 0, (i * 16) + 15, 15, 4 DQBline 2, (i * 16) + (15-i), 15, (i * 16) + i, 0, 40 DQBline 2, (i * 16) + 15, i, (i * 16), (15 - i), 40 NEXT i DQBinitVGA</pre>	

```
FOR i = -16 TO 320

    ' Clear layer 1
    DQBclearLayer 1

    ' Find what frame to draw
    Frame = ((i + 16) \ 2) MOD 15

    ' Draw our frame into layer 1
    DQBxPut 2, (Frame * 16), 0, (Frame * 16) + 15, 15, 1, i, 92

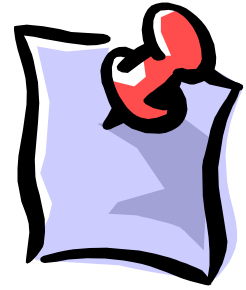
    ' Wait for vertical retrace 1 time
    DQBwait 1

    ' Copy layer 1 onto the screen
    DQBcopyLayer 1, VIDEO

NEXT i

' End the program
DQBclose
END
```

Library Constants



To simplify the use of certain functions, the file DIRECTQB.BI contains some useful constants. Here is the list:

Constant	Value	Can be used by
TRUE	-1	General purpose, no specific function
FALSE	0	General purpose, no specific function
VIDEO	0	All the graphical functions that refer to a layer
B0	&H8000	All graphical functions to refer to 1st base layer
B1	&H8001	All graphical functions to refer to 2nd base layer
B2...B9	&H8002 to &H8009	All graphical functions to refer to base layers 3-10
BSV	0	DQBsaveLayer to specify to save in BSAVE format
BMP	1	DQBsaveLayer to specify to save in BMP format
PCX	2	DQBsaveLayer to specify to save in PCX format
HOR	1	DQBmPut to draw horizontally flipped sprites
VER	2	DQBmPut to draw vertically flipped sprites
BIT.AND	1	DQBtPut to use AND bitwise operation between pixels
BIT.OR	2	DQBtPut to use OR bitwise operation between pixels
BIT.XOR	3	DQBtPut to use XOR bitwise operation between pixels
BOX	0	DQBsetCollideMethod to set bounding box check
PIXEL	1	DQBsetCollideMethod to set pixel-perfect check
FORWARD	0	DQBpalRotate to rotate palette forward
BACKWARD	1	DQBpalRotate to rotate palette backward
CENTERED	&H8000	DQBprint and DQBprints to center text on the screen
NONE	0	DQBsetTextStyle , DQBprints to specify no style
SOLID	1	DQBsetTextStyle , DQBprints to specify solid style
BOLD	2	DQBsetTextStyle , DQBprints to specify bold style
ITALIC	4	DQBsetTextStyle , DQBprints to specify italic style
UNDERLINED	8	DQBsetTextStyle , DQBprints to specify underlined style

(continued on next page)

(continued from previous page)

Constant	Value	Can be used by
BLENDED	16	DQBsetTextStyle , DQBprints to specify blended style
TEXTURED	32	DQBsetTextStyle , DQBprints to specify textured style
KEYANY	-1	DQBplayFLI , DQBwaitKey to specify any key
KEYESC	1	DQBkey to check the ESC key (see Appendix B)
KEYENTER	28	DQBkey to check the ENTER key (see Appendix B)
KEYSPACE	57	DQBkey to check the SPACE key (see Appendix B)
KEYUP	72	DQBkey to check the up arrow key (see Appendix B)
KEYDOWN	80	DQBkey to check the down arrow key (see Appendix B)
KEYLEFT	75	DQBkey to check the left arrow key (see Appendix B)
KEYRIGHT	77	DQBkey to check the right arrow key (see Appendix B)
UP	0	DQBjoyMove to check if the joystick is moved up
DOWN	1	DQBjoyMove to check if the joystick is moved down
LEFT	2	DQBjoyMove to check if the joystick is moved left
RIGHT	3	DQBjoyMove to check if the joystick is moved right
JOY1	0	All the joystick routines to refer to joystick 1
JOY2	1	All the joystick routines to refer to joystick 2
GAMEPAD	2	All the joystick routines to refer to a 4-button gamepad
BUTA	0	DQBjoyFire to check if joystick button A is pressed
BUTB	1	DQBjoyFire to check if joystick button B is pressed
BUTC	2	DQBjoyFire to check if gamepad button C is pressed
BUTD	3	DQBjoyFire to check if gamepad button D is pressed
AUTO	-1	DQBinstallSB to autodetect the soundcard settings
ONCE	0	DQBplaySound to play a sound once
LOOPED	1	DQBplaySound to play a sound repeatedly
ATTRIB.R	&H1	DQBdir\$ attribute to search for read-only files
ATTRIB.H	&H2	DQBdir\$ attribute to search for hidden files
ATTRIB.S	&H4	DQBdir\$ attribute to search for system files
ATTRIB.L	&H8	DQBdir\$ attribute to search for volume labels
ATTRIB.D	&H10	DQBdir\$ attribute to search for directories
ATTRIB.A	&H20	DQBdir\$ attribute to search for archive files

Note The prefix "&H" before the constant value denotes that the value is represented in hexadecimal notation ("hex") rather than decimal.

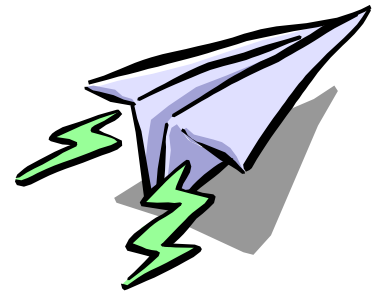
Keyboard Scan Codes



To check the state (pressed or released) of a certain key with the **DQBkey** function, you must pass its keyboard scan code as a parameter. For the most commonly used keys, there are some constants that can help you, so check out Appendix A. Here is the complete list of all the key scan codes (please note that a single scan code can represent more than a single key):

Scan code	Key	Scan code	Key	Scan code	Key
1	ESC	30	A	59	F1
2	1 or !	31	S	60	F2
3	2 or @	32	D	61	F3
4	3 or #	33	F	62	F4
5	4 or \$	34	G	63	F5
6	5 or %	35	H	64	F6
7	6 or ^	36	J	65	F7
8	7 or &	37	K	66	F8
9	8 or *	38	L	67	F9
10	9 or (	39	; or :	68	F10
11	0 or)	40	' or "	69	NUM LOCK
12	- or _	41	` or ~	70	SCR LOCK
13	= or +	42	L SHIFT	71	7 or HOME
14	BACKSPC	43	\ or	72	8 or UP
15	TAB	44	Z	73	9 or PGUP
16	Q	45	X	74	-
17	W	46	C	75	4 or LEFT
18	E	47	V	76	5
19	R	48	B	77	6 or RIGHT
20	T	49	N	78	+
21	Y	50	M	79	1 or END
22	U	51	, or <	80	2 or DOWN
23	I	52	. or >	81	3 or PGDN
24	O	53	/ or ?	82	0 or INSERT
25	P	54	R SHIFT	83	. or DEL
26	[or {	55	* OR PRT SC	87	F11
27	] or }	56	ALT (L & R)	88	F12
28	ENTER	57	SPACE		
29	CTRL (L & R)	58	CAPS LOCK		

Library File Formats



Palette Data

The palette parameter used by **DQBsetPal**, **DQBgetPal**, **DQBfadeIn**, **DQBloadImage**, and **DQBsaveImage** must be a string of 768 characters. This string holds the hue data for each of the 256 colors: three bytes per color, each representing the red, green, and blue intensities ranging 0-63.

If you want to know the green intensity of color 45 of the palette stored in the Pal string, here is the simple code:

```
green = ASC(MID$(Pal, ((45 * 3) + 2), 1))
|
|
| \ Change this with 1, 2, or 3 to
|   get the red, green, or blue hues
|
| \ This is the color number; let's multiply
|   it by 3
```

PAL File Format

Offset	Length	Description
0	768	Palette data string as described above

Mouse Cursor

When calling **DQBsetMouseShape**, you must pass a string of 64 characters as the third parameter; this represents the cursor shape data.

Each cursor shape is a 16x16 pixel bitmap, composed of two bit masks: the "screen mask" and the "cursor mask". The data is stored so that the first 32 bytes of our string are the screen mask, and the last 32 bytes are the cursor mask. Two bytes (16 bits) make a cursor line, so we need 64 bytes to define the two 16x16 pixel masks.

The screen mask specifies which pixels of the cursor are transparent and which are not. A bit set to 0 means that the pixel is transparent, while a bit set to 1 means that the pixel is not transparent (either black or white). The cursor mask specifies what

color each pixel is. If the screen mask is transparent, a bit set to 0 means no color, and a bit set to 1 means that the background color is inverted. If the screen mask is not transparent, a bit set to 0 means the pixel is black, while a bit set to 1 means the pixel is white.

By combining these two masks, the cursor is drawn on the screen. Here are the possible combinations, with their results on the screen:

Bit on screen mask	Bit on cursor mask	Pixel drawn
0	0	Black
0	1	White
1	0	Transparent
1	1	Inverted

This might be a little confusing, so here is an example. The data below represents the default DirectQB mouse cursor (an arrow).

```

&HFF,&H3F,&HFF,&H1F,&HFF,&H0F,&HFF,&H07      \
&HFF,&H03,&HFF,&H01,&HFF,&H00,&H7F,&H00          |
&H3F,&H00,&H7F,&H00,&HFF,&H0F,&HFF,&HBF          |--- Screen mask
&HFF,&HFF,&HFF,&HFF,&HFF,&HFF,&HFF,&HFF          /
&H00,&H00,&H00,&H40,&H00,&H60,&H00,&H70          \
&H00,&H78,&H00,&H7C,&H00,&H7E,&H00,&H7F          |
&H80,&H7F,&H00,&H70,&H00,&H40,&H00,&H00          |--- Cursor mask
&H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00          /

```

Let's examine the bytes of our masks. Remember that two bytes (16 bits) make up one line of a cursor.

Screen mask	Cursor mask
&HFF,&H3F = 0011111111111111	&H00,&H00 = 0000000000000000
&HFF,&H1F = 0001111111111111	&H00,&H40 = 0100000000000000
&HFF,&H0F = 0000111111111111	&H00,&H60 = 0110000000000000
&HFF,&H07 = 0000011111111111	&H00,&H70 = 0111000000000000
&HFF,&H03 = 0000001111111111	&H00,&H78 = 0111100000000000
&HFF,&H01 = 0000000111111111	&H00,&H7C = 0111110000000000
&HFF,&H00 = 0000000011111111	&H00,&H7E = 0111111000000000
&H7F,&H00 = 0000000001111111	&H00,&H7F = 0111111100000000
&H3F,&H00 = 0000000000111111	&H80,&H7F = 0111111110000000
&H7F,&H00 = 0000000000111111	&H00,&H70 = 0111000000000000
&HFF,&H0F = 0000111111111111	&H00,&H40 = 0100000000000000
&HFF,&HBF = 1011111111111111	&H00,&H00 = 0000000000000000
&HFF,&HFF = 1111111111111111	&H00,&H00 = 0000000000000000
&HFF,&HFF = 1111111111111111	&H00,&H00 = 0000000000000000
&HFF,&HFF = 1111111111111111	&H00,&H00 = 0000000000000000
&HFF,&HFF = 1111111111111111	&H00,&H00 = 0000000000000000

The hotspot is passed to **DQBsetMouseShape** in pixel units. A hotspot is the exact pixel of the cursor that the mouse coordinates are at. The hotspot must stay on the

screen all the time, while other parts of the cursor may be off the edge. A hotspot of (0,0) represents the upper-left corner of the cursor bitmap. The hotspot coordinates can range from (-16, -16) to (16, 16).

Since creating a mouse cursor is not an easy process, DirectQB includes a mouse cursor editor in the DirectQB Tools program. It can create and save mouse cursors for use in your own programs.

CUR File Format

Offset	Length	Description
0	32	Screen mask
32	32	Cursor mask

Font Data

A font is contained in a string of 2305 characters. Each font holds the graphical data for 256 characters (2048 bytes), the width in pixels for each character (256 bytes), and the height of all the characters (1 byte).

One character is made up of 8 bytes. Since there are 8 bits in a byte, each row of pixels in a character is made up of 1 byte. This means that the maximum character size is 8x8 pixels.

Each bit of the 8 bytes of graphical data of a character represents the state of a pixel. If the bit is set to 1, the pixel is on (solid). If the bit is set to 0, the pixel is off (transparent or the background color).

Here is the data for a sample character:

```
00010000 = &H10
00111000 = &H38
01101100 = &H6C
01101100 = &H6C
11000110 = &HC6
11111110 = &HFE
11000110 = &HC6
00000000 = &H00
```

As you can see, this character is the shape of an "A".

The pixels off will be transparent if displaying text in transparent mode, otherwise they will be set to the current text background color (set by **DQBsetTextBackCol**).

Note Old fixed-sized fonts are no longer supported by **DQBsetFont**. DirectQB Tools can be used to create fonts that are compatible with DirectQB. Since DirectQB Tools can load fixed-sized fonts, it can be used to convert these fonts to the new format.

FNT File Format

Offset	Length	Description
0	2048	Character data (8 bytes for each character * 256 characters)
2048	256	Width in pixels for each character (1 byte * 256 characters)
2304	1	Height of all the characters

Blender Map Data

Creating a blender map is often a slow process. For this reason, DirectQB includes the **DQBloadBMap** and **DQBsaveBMap** functions. This is the file format that these functions use:

BLN File Format

Offset	Length	Description
0	4	Blender map file format ID string: must be "BMap"
4	1	First mapped foreground color
5	1	Last mapped foreground color
6	256 x n	n blender map chunks, where n is the total number of mapped foreground colors

A blender map chunk is a 256 byte chunk that contains all the 256 possible background color combinations for a foreground color.

Datafile (BIN) File Format

DirectQB datafiles have a somewhat complex structure. The first 1032 bytes of the file make up the header:

Offset	Length	Description
0	7	Datafile file format ID string: must be "DQBPaCk"
7	1	Number of packets used
8	1024	256 packet offset info chunks. Each of these is a 4 byte long integer, specifying the absolute address of the packet data in the file

The packet data follows next. Each packet type has its own small header, and all the packets are encoded and encrypted using the same algorithm.

The method used to encode the packets is a simple RLE (run-length encoding) scheme. Every time a byte has to be repeated, three bytes are inserted: A marker byte (=255), the number of times to repeat the character (up to 255), and then the character to repeat. If a single 255 byte needs to be encoded, the bytes 255, 1, and 255 are inserted.

The encryption uses a simple shift algorithm to encrypt the data packets. To decrypt the packets, the correct password must be used. Before RLE-decoding the data stream, the password bytes are "subtracted" from the encoded data, wrapping the password bytes as needed. For example, if the password is 4 bytes long, and the data to decrypt (before RLE-decoding) is 16 bytes long, Byte 1 of the password would be subtracted from bytes 1, 5, 9, and 13 of the data. Needless to say, when no password is specified, this process has no effect. The password is not stored in the datafile itself, so it is important to remember the password, or you may lose your data! Although this is a simple algorithm, it works quickly and is difficult to decrypt without the correct password.

The following is an explanation of each of the different types of packets that can be stored in a datafile. The structures are accurate only if the packet has been properly decrypted and decoded.

Font Packet

Offset	Length	Description
0	4	Packet ID: must be "FNT" + CHR\$(254)
4	2305	Font data

Image Packet

Offset	Length	Description
0	4	Packet ID: must be "IMG" + CHR\$(254)
4	2	Image width in pixels
6	2	Image height in pixels
8	?	Image data

Sound Packet

Offset	Length	Description
0	4	Packet ID: must be "SND" + CHR\$(254)
4	4	Sound length in bytes
8	?	Sample data

Palette Packet

Offset	Length	Description
0	4	Packet ID: must be "PAL" + CHR\$(254)
4	768	Palette data

Blender Map Packet

Offset	Length	Description
0	4	Packet ID: must be "BMA" + CHR\$(254)
4	1	Number of mapped foreground colors
5	?	Blender map data

Cursor Packet

Offset	Length	Description
0	4	Packet ID: must be "CUR" + CHR\$(254)
4	64	Cursor data

User Data Packet

Offset	Length	Description
--------	--------	-------------

0	4	Packet ID: must be "USR" + CHR\$(254)
4	4	Data length
8	?	<i>Any data</i>

BSAVE File Format

The **BSAVE** file format is the format that QuickBasic uses to save and load a memory image when you use the **BSAVE** or **BLOAD** commands. DirectQB can also save and load files in this format, so you can use your images with any QuickBasic program (you don't need DirectQB).

Since the **BSAVE** format does not directly support a palette, DirectQB appends the palette to the file. This is *not* the same as the Super BSave (SBS) or BS2 format! SBS and BS2 physically place the palette in video RAM, then **BSAVE** 64,768 bytes (so that the palette gets saved). DirectQB appends the palette after the file has been **BSAVED**. To make an SBS or BS2 file compatible with DirectQB, change the length setting (2 bytes at offset 5) from 64768 to 64000. To make a full-screen DirectQB **BSAVE** file compatible with SBS or BS2, change the length setting in the file header from 64000 to 64786.

When DirectQB saves a 320x200 image in the **BSAVE** format, it will be saved just like the QB command:

```
DEF SEG = layerseg
BSAVE imagefile, 0, 64000
```

So you'll be able to load it on the screen without requiring an intermediate buffer. On the other hand, if you use the **BSAVE** format to save an image that is smaller than 320x200, it will be like if you used:

```
GET (x1, y1) - (x2, y2), array
DEF SEG = VARSEG(array)
BSAVE imagefile, VARPTR(array), length
```

BSV File Format

Offset	Length	Description
0	2	File ID: must be CHR\$(253)
1	2	Original segment
3	2	Original offset
5	2	Length of memory image
7	?	Memory image
?	768	Palette data, in the RGB format described previously

Additional File Formats

Aside from the DirectQB-only formats (PAL, CUR, FNT, BLN, BIN, BSV), DirectQB also uses some additional file formats that are fairly common and used by other applications (BMP, PCX, FLI, WAV). If you would like information regarding these file formats and how the data is stored, check out <http://www.wotsit.org>. This is a site with a massive list of file format descriptions, and is very complete.

Known Bugs



I have worked hard to fix all the possible bugs in DirectQB. However, I cannot guarantee that it is totally bug-free.

The main "bug" is a problem with the mouse initialization when calling **DQBinit**. On a K6-200 test machine, **DQBinit** occasionally locked up when attempting to initialize the mouse. On a notebook test machine with an external mouse (Logitech), this problem did not occur. Because of this, I suspect there is a problem with the mouse driver on the K6-200 and not in DirectQB.

There is another small bug that occurs under Windows 95. When setting the mouse cursor hotspot under Windows 95 with **DQBsetMouseShape**, you must multiply the X hotspot coordinate by 2. This is strange because when running the same code in MS-DOS mode, this problem does not occur. I am not sure whether or not this happens in Windows 98/2000/Me.

Additional Bugs Reported by Excalibur North

There is another bug in the **DQBunpackSound** function. After a sound has been loaded from a data file, it will not play.

DQBloadRawSound has another bug that causes only the first part of the sound to be played if the sound data is located after n bytes.

The **DQBnumDrives** function returns an incorrect number of drives under Windows 2000 (3,604 too many). Excalibur North has source for an alternate function that you can use if you want your programs compatible with Windows 2000:

<http://www.excalnor.com/files/drives.bas>.

These are all the bugs I have found in DirectQB. If you find others, please let me know, and I'll try to fix them (See "Credits and Final Words" for contact information).

Version History



Version	Description
Version 1.0 August 30, 1998	First version of DirectQB, with no sound support, a mouse cursor editor, plus a demo of the library. DQBget and DQBput use a custom sprite data format, which is not compatible with standard GET and PUT .
Version 1.1 October 3, 1998	After a month of waiting, version 1.1 brings limited sound support: only one sound effect can be played at once. Sounds are stored and played directly from EMS; the user just calls a function to load and another one to play them. DQBget and DQBput are now compatible with GET and PUT , and DQBput has been optimized a lot. Also added an even faster DQBfastPut , plus the DQBscaledPut routine. Other functions here and there have been added, as well as an in-progress routine to draw gouraud-shaded triangles.
Version 1.11 October 5, 1998	The sound engine bug has been fixed! Now you can play up to 8 sound effects simultaneously! Also added customizable volume setting for each of the eight voices.
Version 1.2 October 27, 1998	Added a lot of new routines. DirectQB now supports rotation and zooming effects, as well as color blending. With the new blender map system, you can now create virtually any color effect. Also fixed some bugs here and there, and added some extra sound handling routines.
Version 1.3 November 15, 1998	Mainly internal enhancements. Added an int 24h handler for critical DOS errors. DQBxPut draws a sprite without using external QB arrays, saving memory. New faster DQBprint function now supports non-fixed sized fonts, as well as different print styles (normal, bold, italic, or underlined). DirectQB has its first 3D functions! DQBtri and DQBgtri draw flat-shaded and gouraud-shaded triangles very quickly.

(continued on next page)

(continued from previous page)

Version	Description
Version 1.31 November 23, 1998	Finally added texture mapped triangle support. Most of the graphical functions have been accelerated a little, plus other minor changes.
Version 1.4 December 18, 1998	No more DQBlite! Now the entire library is split into 15 object files, so you can build your own custom version of DQB, with only the functions that you need! Added color blending support to all the triangle functions, a flipped put routine, and support for layers in base memory. In addition, from this release on, the library will be distributed without executables or the LIB and QLB files. A utility is included that builds the LIB and QLB files.
Version 1.41 December 22, 1998	Small bugs fixed, and the new DQBpaint function added.
Version 1.42 December 29, 1998	Another maintenance release, with lot of small bugs fixed. Base layers are now automatically allocated in base memory, and they finally work. Also added two new joystick routines, and the MAKEDQB utility has been almost recoded to work more reliably and provide for easier modifications.
Version 1.5 March 25, 1999	Major changes! Sound engine now supports sound resampling! Added the DQBscrollArea function, plus a high precision timer, pixel-perfect collision detection, a fast FLI player, and new font routines (now also with support for textured text!). Also added lots of general purpose functions. The library has been sped up a little, and there are several bug fixes. The DQBMAKE batch utility has been replaced by the user-friendly DirectQB Library Manager program, which allows you to select each module to add to your copy of DQB, and builds an optimized DIRECTQB.BI file for you automatically.
Version 1.51 April 8, 1999	Minor bug fixes, plus the new DQBcollideOnLayer function, and the DQB Library Manager 1.1, with autodetection of your QuickBasic 4.5 directory.

(continued on next page)

(continued from previous page)

Version	Description
Version 1.6 May 29, 1999	Major, and maybe last DirectQB release. Added support for up to 10 blender maps active at the same time, datafiles, improved sound quality, any size image load/save, new layer copy routines, improved font routines, support for Windows long filenames, many bug-fixes and the new DQB DataFile Encoder program. Also, the bug with the DQB Library Manager directory finder has been fixed.
Version 1.61 June 1, 1999	Minor bug fixes, DQBpPut function added, datafile handling routines modified to work with numbers as packet IDs instead of strings.

Inside Library Modules



Here is the list of functions contained in each of the OBJ files that make up the DirectQB library. Every time you use a function, you will need the corresponding module, which will also be linked to your final program executable (EXE). By default, all these modules are included in the library. You can build your own custom set of routines by using different combinations. See Chapter 1, “Accessing the Library”, for details.

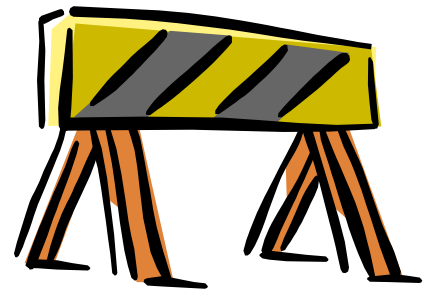
Module	Contains
MAIN.OBJ	DQBinit, DQBver, DQBmapLayer, DQBclose, DQBsort, DQBinitVGA, DQBinitText, DQBsetBaseLayer, DQBcopyLayer, DQBclearLayer, DQBwait, DQBsetFrameRate, DQBframeReady, DQBangle, DQBerror\$, DQBid\$, DQBpeek, DQBpoke
DRAW.OBJ	DQBcopyTransLayer, DQBcopyHitLayer, DQBpset, DQBpoint, DQBline, DQBgline, DQBellipse, DQBbox, DQBboxf, DQBpaint, DQBscroll, DQBscrollArea, DQBsetTransPut, DQBsetSolidPut, DQBget, DQBput
IMAGE.OBJ	DQBloadImage, DQBsaveImage, DQBplayFLI, DQBopenFLI, DQBplayFLIstep, DQBcloseFLI
SPRITE.OBJ	DQBsize, DQBsetClipBox, DQBspPut, DQBbrPut, DQBfpPut, DQBxPut, DQBmPut, DQBhPut, DQBtPut, DQBpPut, DQBputOver, DQBcollide, DQBcollideOnLayer, DQBsetCollideMethod
BIT.OBJ	DQBsetBit, DQBresetBit, DQBreadBit, DQBtoggleBit, DQBshiftLeft, DQBshiftRight
PALETTE.OBJ	DQBsetCol, DQBgetCol, DQBfindCol, DQBfindPalCol, DQBsetPal, DQBgetPal, DQBfadeIn, DQBfadeStepIn, DQBfadeTo, DQBfadeStepTo, DQBpalOff, DQBpalRotate

(continued on next page)

(continued from previous page)

Module	Contains
FONT.OBJ	DQBprint, DQBprints, DQBlen, DQBsetBIOSfont, DQBsetFont, DQBsetTextBackCol, DQBsetTextStyle, DQBsetFontTexture, DQBloadFont, DQBsetTextSpacing, DQBsetTextBMap
DISK.OBJ	DQBdir\$, DQBdrive\$, DQBpath\$, DQBnumDrives, DQBsetDrive\$, DQBchDir
BLENDING.OBJ	DQBfilterBox, DQBbPut, DQBcreateBMap, DQBsetBMap, DQBgetBMap, DQBcopyBlendLayer, DQBloadBMap, DQBsaveBMap, DQBremoveBMap
KEYBOARD.OBJ	DQBinstallKeyboard, DQBremoveKeyboard, DQBkey, DQBreadKey, DQBwaitKey, DQBasc, DQBkey\$
JOYSTICK.OBJ	DQBjoyDetected, DQBpollJoy, DQBjoyX, DQBjoyY, DQBjoyMove, DQBjoyFire, DQBresetJoy, DQBsetJoySens
MOUSE.OBJ	DQBmouseDetected, DQBmouseX, DQBmouseY, DQBmouseLB, DQBmouseRB, DQBsetMousePos, DQBmouseShow, DQBmouseHide, DQBsetMouseRange, DQBsetMouseShape, DQBsetMouseSpeed, DQBresetMouse
SOUND.OBJ	DQBinstallSB, DQBloadSound, DQBloadRawSound, DQBplaySound, DQBinUse, DQBpauseSound, DQBresumeSound, DQBstopVoice, DQBsetVoiceVol, DQBremoveSB, DQBsetVolume
3D.OBJ	DQBtri, DQBgtri, DQBttri, DQBbtri, DQBbgtri, DQBbttri, DQBfttri, DQBsetTextureSize
DATAFILE.OBJ	DQBopenDataFile, DQBunpackImage, DQBunpackSprite, DQBunpackSound, DQBunpackPal, DQBunpackBMap, DQBunpackFont, DQBunpackCursor, DQBunpackUser, DQBcloseDataFile

Error Messages



DirectQB has a built-in error messaging system. This means that whenever a function fails, other than the error code returned by the function itself, an error message is stored in a string. This string can be retrieved at any time by calling **DQBerror\$**. You can use this function to notify the user of exactly which error occurred, without using a custom error handler to handle all the possible DirectQB error messages. The following is an example of how to use this system.

```
' Suppose you want to initialize the library with 7 layers, 5 sounds
' and 120 KB of free EMS memory:
```

```
IF DQBinit(7, 5, 120) THEN
  DQBclose
  PRINT DQBerror$
END
END IF
```

```
' If we get here, DirectQB was successfully initialized
```

If the **DQBinit** call fails (almost all the DQB functions return a non-zero value on errors), **DQBclose** is called to avoid possible conflicts, and then the proper error message is shown to the user, and the program ends. What is stored in **DQBerror\$** depends on the function that failed. The table below contains all the possible error messages.

Error Message	May be caused by
"Library already initialized"	DQBinit
"386 or better CPU not found"	DQBinit
"No EMM driver detected"	DQBinit
"Not enough free EMS memory"	DQBinit
"Unable to open file"	DQBloadImage, DQBplayFLI, DQBloadFont, DQBloadBMap, DQBloadSound, DQBloadRawSound
"General file reading error"	DQBloadImage, DQBplayFLI, DQBloadFont, DQBloadBMap, DQBloadSound, DQBloadRawSound

(continued on next page)

(continued from previous page)

Error Message	May be caused by
"Blender map already allocated"	DQBcreateBMap
"Not enough free base memory"	DQBsetBaseLayer, DQBplayFLI, DQBcreateBMap, DQBinstallSB
"Blender map is not active"	DQBloadBMap, DQBsaveBMap
"Cannot create file"	DQBsaveImage, DQBsaveBMap
"General file writing error"	DQBsaveImage, DQBsaveBMap
"Unknown or bad file format"	DQBloadImage, DQBplayFLI, DQBloadSound
"No sounds initialized"	DQBinstallSB
"DMA channel not supported"	DQBinstallSB
"Failed to reset sound card DSP"	DQBinstallSB
"High mixing speed not supported"	DQBinstallSB
"Old sound card not supported"	DQBinstallSB
"File format not supported"	DQBloadSound
"Sound exceed 64k size limit"	DQBloadSound
"Unknown sound slot"	DQBloadSound, DQBloadRawSound, DQBunpackSound
"Cannot read past end of file"	DQBloadRawSound
"Can't open two FLI files at once"	DQBplayFLI, DQBopenFLI
"Incompatible blender map"	DQBloadBMap, DQBunpackBMap
"BLASTER variable not set"	DQBinstallSB
"Sound engine not yet initialized"	<i>Reserved for future use</i>
"Datafile not yet opened"	DQBunpackFont, DQBunpackImage, DQBunpackSprite, DQBunpackSound, DQBunpackBMap, DQBunpackPal, DQBunpackCursor, DQBunpackUser
"Unknown packet ID"	DQBunpackFont, DQBunpackImage, DQBunpackSprite, DQBunpackSound, DQBunpackBMap, DQBunpackPal, DQBunpackCursor, DQBunpackUser
"Bad password or corrupted data"	DQBunpackFont, DQBunpackImage, DQBunpackSprite, DQBunpackSound, DQBunpackBMap, DQBunpackPal, DQBunpackCursor, DQBunpackUser
"Can't open two datafiles at once"	DQBopenDataFile
"Operation successful"	<i>Any function</i>

Credits and Final Words



DirectQB was created by Angelo Mottola of Enhanced Creations 1998-99.

Special Thanks To

Petter Holmberg	for the original QB version of the core algorithm used for DQBpaint , many other hints, and for his constant help and interest in my work.
Rich Geldreich	for the original QB algorithm used by DQBellipse
Peter Norton	for his great online guide to assembly language
Ralph Brown	for his huge and detailed list of DOS interrupts
The PC Game Programmers' Encyclopedia	as an infinite source of information

...And all of you on the EC wwwboard and on ICQ for your constant support and your many suggestions that bring me to constantly improve DirectQB.

Final Words

If you have any hints, suggestions, or bug reports, please contact me using one of the following methods:

E-mail a.mottola@ecplusplus.com
ICQ 24084401
ICQ pager 24084401@pager.mirabilis.com

Visit the new Enhanced Creations++ website located at:

<http://www.ecplusplus.com>

DirectQB is freeware. The only thing I'm asking you is that you include my name somewhere in the credits section of your game if you use it. Once finished, it would be nice for me to see your work!

Thanks for using DirectQB!

