

Contents



PiXCL v4.4 / 5.1 Command Reference

[Windows 2D Graphics and Imaging Functions](#)

[File and Directory Management](#)

[String Functions](#)

[Math Functions](#)

[Time Functions](#)

[Clipboard Functions](#)

[Menu Functions](#)

[Program Branching and Control Functions](#)

[Screen and Keyboard I/O Functions](#)

[Window Management Functions](#)

[INI file and Registry Access](#)

[MultiMedia Functions](#)

[TWAIN-compliant Device Functions](#)

[IDRISI for Windows™ related commands](#)

[Miscellaneous Functions](#)

[Importing images from TWAIN-compliant devices](#)

[Printing documents and images with PiXCL](#)

[Using the Windows Shell functions with PiXCL](#)

[User Defined Command Extensions](#)

[Error Messages](#)

[More Information](#)

[Detailed Alphabetical Command Reference](#)

[Image Processing Commands](#)

[Internet Technical Support](#)

For Help on Help, Press F1

Overview of PiXCL v4.4 and v5.1 - the graphics and image processing language tool for Windows 9x / ME / NT4 / 2000

[Overview](#)

[Memory Requirements](#)

[Rules and Syntax](#)

[Starting a Program](#)

[Stopping a Program](#)

[The Coordinate System](#)

[Drawing Tools](#)

[Drawing and Writing](#)

[Flow Control Commands](#)

[Getting Mouse Input](#)

[Getting Keyboard Input](#)

[Running Other Programs](#)

[Invoking on-line Help Files](#)

[Managing Files and Directories](#)

[Clipboard Operations](#)

[Creating and using INI files](#)

[Accessing and updating the Registry](#)

[Printing files](#)

[Message Boxes](#)

[Text Boxes](#)

[List Boxes](#)

[ComboBoxes](#)

[Windows 95 and NT 4.0 Common Controls](#)

[Getting a Filename](#)

[Managing Windows](#)

[Pausing a Program](#)

[Building Menus](#)

[Building Toolbars and ToolWindows](#)

[3-D Command Buttons](#)

[Building Runtime .EXE Files](#)

[Creating your own CD-ROMs](#)

[Software License and Limited Warranty](#)

Windows 2D Graphics and Imaging Functions

PiXCL includes a comprehensive set of commands to draw lines, rectangles, circles and ellipses in various colors and pen sizes. There are also polygon flood and fill functions. The listed commands below produce a popup window. If you want full information, use the Help Search options.

General Screen Draw Commands

- [AddFont](#)
- [ChooseColor](#)
- [ChooseFont](#)
- [CustomColor](#)
- [DrawArc](#)
- [DrawChord](#)
- [DrawEdgeRectangle](#)
- [DrawEllipse](#)
- [DrawFlood](#)
- [DrawFloodExt](#)
- [DrawGrid](#)
- [DrawLine](#)
- [DrawNumber](#)
- [DrawFpNumber](#)
- [DrawShadowNumber](#)
- [DrawShadowFpNumber](#)
- [DrawPie](#)
- [DrawPolyCurve](#)
- [DrawPolygon](#)
- [DrawPolyLine](#)
- [DrawRectangle](#)
- [DrawRoundRectangle](#)
- [DrawShadeRectangle](#)
- [DrawStatusText](#)
- [DrawText](#)
- [DrawShadowText](#)
- [DrawTextExt](#)
- [DrawShadowTextExt](#)
- [DrawTriangle](#)
- [GetTextSpacing](#)
- [RotateRectangle](#)
- [UseBrush](#)
- [UseBrushPattern](#)
- [UseFont](#)
- [RemoveFont](#)
- [SetFontEscapement](#)
- [SetTextSpacing](#)
- [UsePen](#)

BitMap Related Commands

- [CountBitmapColors](#)
- [DrawBitMap](#)
- [DrawIcon](#)
- [DrawIconFile](#)
- [DrawSizedBitMap](#)
- [DrawTrBitmap](#)
- [DrawTrSizedBitmap](#)
- [DrawZoomedBitmap](#)
- [GetBitMapDim](#)
- [GetListBitMapDim](#)
- [InvertRectangle](#)
- [LoadBitmap](#)
- [PrintBitmap](#)
- [RawDataParamBox](#)
- [ReadBitmapRect](#)
- [ReadRawBitmap](#)
- [RemapImage](#)
- [RotateRectangle](#)
- [SetColorPalette](#)
- [SaveBitmap](#)
- [SaveRectangle](#)
- [SetDrawMode](#)
- [ExportHistogram](#)
- [FlashBMWindow](#)
- [SetBMWMouse](#)
- [DrawBMWPoint](#)
- [WriteBitmapRect](#)

Miscellaneous Screen Commands

- [GetBackground](#)
- [GetScreenCaps](#)
- [GetPixel](#)
- [GetScreenWorkArea](#)
- [DrawAnimatedRects](#)
- [DrawBackGround](#)
- [DrawBackgroundRegion](#)
- [DrawCaption](#)
- [DrawFocusRectangle](#)
- [UseBackGround](#)
- [UseCoordinates](#)

File and Directory Management

File and Directory create, exist, move, copy and delete operations are supported. File size and total / free disk space commands are available. You can also read and write system and application INI files.

The listed commands below produce a popup window with a summary of the syntax. If you want full information, use the Help Search options.

| | |
|---------------------------------|----------------------------------|
| DirChange | FileGetSize |
| DirExplore | FileGetTime |
| DirGet | FileMove |
| DirGetSystem | FileName |
| DirGetWindows | FilePath |
| DirListFiles | FileRead_ASCII |
| DirMake | FileRead_Binary |
| DirRemove | FileRead_INI |
| DiskChange | FileRename |
| DragAcceptFile | FileSaveAs |
| GetDragList | FileWrite_ASCII |
| DropFileServer | FileWrite_Binary |
| FileCopy | FileWrite_INI |
| FileDelete | FindExecutable |
| FileExist | GetDiskSpace |
| FileExtension | GetVolumeType |
| FileGet | |
| FileGetDate | |
| FileGetDateExt | |
| FileGetTempName | |

String Functions

The listed commands below produce a popup window. If you want full information, use the Help Search options. All the commonly used string and list handling functions are supported, as follows.

[Ansi](#)

[Chr](#)

[EpStr](#)

[FpVal](#)

[HexToNum](#)

[Instr](#)

[ItemCount](#)

[ItemExtract](#)

[ItemLocate](#)

[ItemInsert](#)

[ItemRemove](#)

[LCase](#)

[Left](#)

[LeftOf](#)

[Len](#)

[NumToHex](#)

[Pad](#)

[Right](#)

[RightOf](#)

[Space](#)

[Str](#)

[StrCmp](#)

[StrCmpl](#)

[StrRepl](#)

[StrReplAll](#)

[StrRev](#)

[Substr](#)

[Trim](#)

[TrimExt](#)

[UCase](#)

[Val](#)

Math Functions

PiXCL supports a floating point as well as integer variables, plus a set of standard math library functions. The four standard math operations plus modulus are provided in command syntax.

[Acos](#)

[Asin](#)

[Atan](#)

[Average](#)

[Cos](#)

[Cosh](#)

[Exp](#)

[Float](#)

[FpAbs](#)

[FpStr](#)

[FpVal](#)

[Hypot](#)

[Int](#)

[Log10](#)

[LogE](#)

[Negate](#)

[Pow](#)

[Random](#)

[Sin](#)

[Sinh](#)

[Sqrt](#)

[Tan](#)

[Tanh](#)

Time Functions

These functions provide access to get and set the local and system time, get the current time zone setting, and return a usable time string to programs. They can also be used for scheduling operations such as running other programs at specific times.

[GetLocalTime](#)

[GetSystemTime](#)

[GetTimeZone](#)

[SetLocalTime](#)

[SetSystemTime](#)

[TimeToASCII](#)

Clipboard Functions

Clipboard Put, Get and Append ASCII data commands are available, plus transferring image data to the clipboard for transfer to other imaging applications.

[ClipboardAppend](#)

[ClipboardEmpty](#)

[ClipboardGet](#)

[ClipboardPut](#)

[TWAIN_AcquireToClipboard](#)

[ClipboardGetBitmap](#)

[ClipboardPutBitmap](#)

[CopyWindowToClipboard](#)

Menu Functions

A Main Menu bar and single level pull down menus are supported. While child menus are not supported in this version of PiXCL, additional menu levels can be added by redrawing the complete menu bar. Each menu item can have as many pull-down items as you require.

[ChangeMenuItem](#)

[GetMenuStatus](#)

[InfoMenu](#)

[SetMenu](#)

[SetPopupMenu](#)

Program Branching and Control Functions

PIXCL supports the basic unstructured [If..Then](#) and [Goto](#) commands, plus the modern structured [If...Else...Endif](#) syntax. Thirty-two levels of embedded [If...Else...Endif](#) structures are supported. Up to Thirty-two levels of embedded level Structured [For-Next](#) and [While-EndWhile](#) loops are available. It is also possible to construct [Do-While](#) and [Do-Until](#) operations with the labels and [Goto](#) statements. For integer variables, the [Switch-Case-EndSwitch](#) structure is provided.

[Gosub](#)

[Goto](#)

[If-Then](#)

[If-Else-Endif](#)

[Return](#)

[For-Next Loops](#)

[While Loops](#)

[Switch Statements](#)

Screen and Keyboard I/O Functions

PIXCL provides a comprehensive set of input and output resources for common, built-in, custom and common dialog boxes, common controls, keyboard and mouse, including support for the Microsoft Intellimouse™ and similar middle mouse button, and the Explorer Mouse™ and similar X1 and X2 buttons..

| | |
|-----------------------------------|-------------------------------------|
| Button | SetMouse |
| DialogBox | SetShftMouse |
| ImageBox | SetCtrlMouse |
| ListBox | SetDbfMouse |
| ListBoxExt | SetRightMouse |
| MessageBox | SetShftRightMouse |
| PasswordBox | SetCtrlRightMouse |
| TextBox | SetDbfRightMouse |
| TextBoxExt | SetMidMouse |
| SetEditControl | SetShftMidMouse |
| ComboBox | SetCtrlMidMouse |
| ProgressBar | SetDbfMidMouse |
| UpdateProgressBar | SetDrawMouse |
| StatusWindow | SetWaitMode |
| DrawFrameControl | Toolbar |
| DrawStatusWinText | GetToolbarBtnStatus |
| | ChangeToolbarBtn |
| ReportMouse | CustomizeToolBtn |
| SetEditControl | UseCursor |
| SetKeyboard | WaitInput |
| SetMenu | PXLResume |
| SetPopupMenu | PXLresumeAt |
| | GetCopyDataMsg |
| | SendCopyDataMsg |

Serial I/O Functions

PiXCL includes support for simple communications using the serial ports COM1 - COM4. This support is designed to enable devices such as digitizing pads and tables that output coordinate data strings, and accept control commands.

[ClearCommPort](#)

[GetCommPort](#)

[EscCommFunction](#)

[ReadCommPort](#)

[SetCommPort](#)

[WaitCommEvent](#)

[WriteCommPort](#)

Window Management Functions

PiXCL includes a set of commands to manage the appearance and size of any window.

[EnumWindows](#)

[EnumChildWindows](#)

[SendKeys](#)

[SetSendKeysPriority](#)

[SetPriority](#)

[SetWindow](#)

[ProgressBar](#)

[UpdateProgressBar](#)

[StatusWindow](#)

[DrawStatusWinText](#)

[UseCaption](#)

[WinAdjustRect](#)

[WinClose](#)

[WinExist](#)

[WinGetActive](#)

[WinGetLocation](#)

[WinLocate](#)

[WinHelp](#)

[WinHTMLHelp](#)

[WinSetActive](#)

[WinShow](#)

[WinTitle](#)

INI File and Registry Access Functions

PIXCL provides full read and write access to both initialization files and the Windows registration database or Registry.

[FileRead_INI](#)

[FileWrite_INI](#)

[RDBCcloseKey](#)

[RDBCreateKey](#)

[RDBDeleteKey](#)

[RDBEnumKey](#)

[RDBOpenKey](#)

[RDBQueryKey](#)

[RDBQueryValue](#)

[RDBSetValue](#)

Multimedia Functions

If you have a SoundBlaster™ compatible sound card installed in your PC, you can use these commands to play sounds and control the card. Please note that **not all cards support all functions**. If a sound card is not installed in your PC, these commands are ignored. If you are uncertain if your WAV play card is supported, try the first two commands in the list below and see what capabilities are reported.

[WAVGetDevCaps](#)

[WAVGetNumDevs](#)

[WAVGetPitch](#)

[WAVGetPlayRate](#)

[WAVGetVolume](#)

[WAVPlaySound](#)

[WAVSetPitch](#)

[WAVSetPlayRate](#)

[WAVSetVolume](#)

TWAIN-compliant Device Functions

PIXCL provides access to any TWAIN-compliant image import device, such as scanners and digital cameras. All commands start with "TWAIN_" so they can be easily identified in a program source. See [Importing images from TWAIN-compliant devices](#) for more detailed information.

| | |
|--|--|
| TWAIN_AcquireNative | TWAIN_Getstate TWAIN_IsAvailable |
| TWAIN_AcquireToClipboard | TWAIN_LoadSourceManager |
| TWAIN_AcquireToFilename | TWAIN_OpenDefaultSource |
| TWAIN_CloseSource | TWAIN_OpenSourceManager |
| TWAIN_CloseSourceManager | TWAIN_PxlVersion |
| TWAIN_DisableSource | TWAIN_SelectSource |
| TWAIN_EnableSource | TWAIN_SetBitDepth |
| TWAIN_GetBitDepth | TWAIN_SetCurrentRes |
| TWAIN_GetBitmapParams | TWAIN_SetCurrentUnits |
| TWAIN_GetCurrentRes | TWAIN_SetPixelFormat |
| TWAIN_GetCurrentUnits | TWAIN_UnloadSourceManager |
| TWAIN_GetPixelFormat | |

IDRISI for Windows related commands

The IDRISI GIS, copyright Clark University in Massachusetts, is one of the most popular geographic information systems, world wide. PiXCL versions provide direct access to the IDRISI environment to make development of additional applications using PiXCL a relatively simple process.

Note very importantly that a licensed version of IDRISI for Windows v2.x must be running on the system and the API DLL's present to make use of the API library functions, otherwise these commands do nothing. Functions in this group include:

[IDR_CloseIdrisi](#)
[IDR_GetDataDir](#)
[IDR_GetDir](#)
[IDR_GetExtensions](#)
[IDR_GetLanguage](#)
[IDR_GetProgress](#)
[IDR_InitProgressTracking](#)
[IDR_IsPresent](#)
[IDR_Launch](#)
[IDR_LaunchModule](#)
[IDR_RegisterClient](#)
[IDR_SetDataDirectory](#)
[IDR_SetDebugMode](#)
[IDR_SetExtensions](#)
[IDR_SetProgress](#)
[IDR_UnRegisterClient](#)

Miscellaneous Functions

[AbortShutDown](#)
[AboutPiXCL](#)
[AboutUser](#)
[AppWindowHandle](#)
[AutoProgressBar](#)
[Beep](#)
[End](#)
[ExitWindows](#)
[FreeBitMap](#)
[FreeBitMapAll](#)
[FreeVar](#)
[FreeVarAll](#)
[GetCmdLine](#)
[GetCPUInfo](#)
[GetEnvString](#)
[GetEnvVariable](#)
[GetFontFace](#)
[GetScreenCaps](#)

[GetPixel](#)
[GetSystemMetrics](#)
[GetSysPowerStatus](#)
[GetTempPath](#)
[LogOff](#)
[ListLoadedBitmaps](#)
[MessageBeep](#)
[PrintFile](#)
[SetROPcode](#)
[SetEnvVariable](#)
[Set_Variable](#)
[ShellAbout](#)
[Shutdown](#)
[WinVersion](#)

User Defined Commands Extension

An **optional** component for PiXCL 5.0 and later, and included in geoPiXCL, is a Programmer's API to support User Defined Commands. This API contains all the necessary information to access the internal data structures of PiXCL and geoPiXCL, and includes a Visual C/C++ 6 sample project that demonstrates how extension commands are implemented.

User defined commands can provide more image processing functions, new dialogs and other resources, and act as a bridge into other third party DLLs.

To purchase the API, please contact [VYSOR Integration Inc.](#) or go to the purchase link on our Web pages.

More Information

This section is for knowledgeable Windows programmers who want more information on the internals of PiXCL and PiXCL runtime programs. **Most readers can skip this section.** Windows internal programming terms are not defined, as technically knowledgeable readers are expected to understand them.

Firstly, please note that [PiXCL50.exe](#), and [PXL_make50.exe](#) are a matched set by version and build number. [PXL_make44](#) cannot use 16-bit versions of PiXCL or any versions of PiXCL 4.2 or earlier. In addition, the PXLimage.dll has a version number as well. Several builds of PiXCL will use the same DLL version (e.g. PiXCL 4.10 – 5.0). When we update the DLL, we issue a new version of PiXCL with the new DLL.

PiXCL is a simple interpreted language, hence a PiXCL script will not execute as fast as the equivalent application written in C or C++ with a Windows compiler, but the development time is very significantly less, because all the complexity of Windows is hidden away in the interpreter. Most of the PiXCL coding you will write is related to presenting a user interface, and the results of image processing library functions. The PXLimage.DLL is written in C, and provides fast processing of command functions when speed is required. In developing an application, however, the tool must match the job. If you find that the capabilities of PiXCL are inadequate for your application, we suggest that MS-Visual C/C++™, MS-Visual Basic™, or Borland Delphi™ would be better choices for development software, even though the development time is often significantly greater. The PiXCL Image Processing Library API is an available product.

A PiXCL application does not consume much of the system resources. According the Windows resource meter, when PiXCL is running it uses 2-3 % of system, 1-2 % of user and 10-12 % of GDI resources. By way of comparison, MS-Word 7 requires 6-10% of both system and user, and 10-12 % of GDI resources.

[PiXCL](#) accepts command line arguments. If you are testing a script e.g. with a [PiXCL50 script.pxl](#) commandline, the first argument is expected to be a PiXCL script filename. This is the only argument that is accepted in this case, because Windows will accept filenames with embedded spaces.

Additional command line arguments are supported by a PiXCL runtime, space delimited, and be can accessed with the [GetCmdLine\(...\)](#) command.

PiXCL first checks if there is an attached script by looking for a string at a specific internal location. If the string is not found, it either accesses the argument file (if any), or prompts for a script file.

Next, PiXCL checks for correct syntax in the target script, and displays an error message dialog box with the offending line of code displayed. This will usually be the location of the error, unless the error is a missing } or “, as these characters are used to delimit comments and strings. That is, the error may be offset in the file. If the error line syntax looks correct, look further back into the file.

If everything is correct, the PiXCL window classname is registered with Windows, and the script is executed.

What PiXCL does is create a memory and screen display context. These are based on a bitmap created the current size and pixel depth of the Windows display, e.g. 640x480, 1024x678, 1280x1024 or larger, and at 8, 16, 24 bits or 32 per pixel. All paint and draw commands make changes in this bitmap, which then appear on your screen. It is possible to write only the screen display context by using the [SetDrawMode](#) and [SetROPcode](#) commands with appropriate arguments.

[PXL_make50](#) is the Runtime builder, and accepts command line arguments. Under user control, it takes the PiXCL interpreter, verifies that is the correct version by reading several binary signatures in the PiXCL50.exe file, then combines it with the encrypted specified script into a 32-bit Windows EXE file. It also sets a double word in the final binary so that the interpreter knows that it has a script appended.

If you have access to a Resource Compiler tool you have the ability to modify the PiXCL interpreter resources. This is not recommended, as it will change the size and binary composition of the interpreter such that the runtimes will not be able to access an embedded script. The runtimes will likely still function, but only as a PiXCL interpreter that requests a script file. Remember, [PiXCL50.exe](#) and [PXL_make50.exe](#) are a matched set by version.

Any modification the PiXCL binaries other than replacement of icons (dual mode 32x32 plus 16x16 16-color only) is also an infringement of your user license agreement.

Suggestions for improvements and bug reports (hopefully a rare occurrence) are welcomed. Please contact [YYSOR Integration Technical Support](#).

Error Messages

Debugging Scripts

Error messages from the PiXCL interpreter are listed alphabetically below, with the typical cause and solution. In this version of PiXCL, most reported errors are severe enough to be fatal and will cause the program to exit. This is usually because there is a syntax error, or much less commonly, not enough system resources available. This can occur if you are running many programs at once, more so with Windows 95/98 than Windows NT. Rebooting Windows will usually cure the problem. In some cases, you can allow for the possible error condition in your script so that the program does not crash. Some errors are non-fatal (e.g. when you try to Run(...) a program that does not exist), and display a MessageBox, then continue.

Error messages are listed below in alphabetical order.

AbortShutdown failed.

Cause: Your NT system would not allow the command to be executed, probably because the required privileges were not set.

Solution: Try using the RunExt() command with the Shutdown command in the new script.

ANSI code must be from 0 to 255.

Cause: You used a code greater than 255 in the Chr() command.

Solution: Use a code in the range 0 - 255.

AdjustTokenPrivileges enable failed.

Cause: Windows NT has a problem with a Shutdown, AbortShutdown or ExitWindows command. Appropriate privileges have not been set.

Solution: Issue the command from a script started with the RunExt() command.

Cannot access Clipboard. Another application has prevented access.

Cause: Your script cannot write or read ascii data from the ClipBoard. This would be most unusual. Some packages make extensive use of the Clipboard.

Solution: Try using the ClipBoardEmpty command. Shutdown a few other applications.

Cannot locate the PiXCLmsg.dll file.;

Cause: The file is not in the PiXCL installation directory.

Solution: Locate the file or re-install PiXCL.

Cannot open the file.

Cause: An unreadable file (e.g. an EXE file) has been used as a PiXCL argument.

Solution: Check the contents of the argument file.

Cannot read the PXL script file.

Cause: An unreadable file (e.g. an EXE file) has been used as a PiXCL argument.

Solution: Check the contents of the argument file.

Cannot read the bitmap file.

Cause: A DrawBitmap command has tried to access an unrecognized image format.

Solution: Verify the image can be displayed, or convert the image to a known format.

Cannot run program.

Cause: Non-Fatal Error. A MessageBox appears with on of the following messages.

"Reported Cause: Out of Memory or Resources.;"

"Reported Cause: EXE File not found.;"

"Reported Cause: PATH not found.;"

"Reported Cause: Bad EXE format.;"

Solution: Check the format of the EXE file. An NT binary for another cpu type (e.g. MIPS or Alpha) will cause this error.

Can't send keys; try increasing PauseRespond parameter.

Cause: The SendKeys command is having problems communicating with another application.
Solution: Adjust your script.

Color value greater than 255.

Cause: You specified an illegal value in a Draw*() command or UseFont() command.
Solution: Values must be in the range 0 - 255.

Could not load RLE file.

Cause: A DrawBitmap command has tried to access an unrecognized image format.
Solution: Verify the image can be displayed, or convert the image to a known format.

CreatePalette() failed.

Cause: The system was unable to create a palette with the SetPalette90 command.
Solution: Check system resources. It may be necessary to reboot Windows.

Delimiter must be a valid character

Cause: Error in the ListBox() command.
Solution: Use an ANSI character in the range 0 - 255. The most common characters used are 'space', 'colon (:)', 'semi-colon' and 'pipe (|)'.
Solution: Use an ANSI character in the range 0 - 255. The most common characters used are 'space', 'colon (:)', 'semi-colon' and 'pipe (|)'.

Divide by zero

Cause: Error in a math operation
Solution: Ensure that the divisor is 1 or greater.

File is not valid bitmap format.

Cause: You have tried to load a bitmap that is not BMP or RLE format with DrawBitMap() or DrawSizedBitMap().
Solution: Convert the image file to BMP or RLE format.

Filter must contain pairs of filter elements.

Cause: The FileGet() command filter is incorrect. The error is being passed back to PiXCL from the COMMDLG.DLL supplied with Windows.
Solution: Correct the filter syntax.

Fractional number.

Cause: A fractional number was entered. PiXCL 4.4 supports integers only.
Solution: Use an integer.

Invalid keystrokes argument.

Cause: The SendKeys command has located an invalid key sequence.
Solution: Check the sequence in your script.

Invalid number.

Cause: You entered a string that was not an integer.
Solution: Enter a valid integer.

Invalid repetition count in keystrokes argument.

Cause: The SendKeys command has located an invalid argument.
Solution: Check the argument in your script.

Label is multidefined.

Cause: A common error. There are two or more LABELs which are the same.
Solution: Ensure that all labels are unique.

Label not found.

Cause: PiXCL cannot find a LABEL referred to in a script command such as GoTo, SetMenu, SetKeyBoard or SetMouse.
Solution: Put in the missing LABEL. The LABEL may also not be on the beginning of a line, or may start with a number.

MessageBox process failed: unsupported Button code.

Cause: A MessageBox command has used an invalid button code. Acceptable codes are 1, 2 or 3.
Solution: Correct your script.

Not enough memory.

Cause: A general purpose error that appears if a system call has failed through lack of memory.
Solution: Try shutting down unnecessary applications. If the problem persists, contact VYSOR Integration Inc Technical Support.

Not a legitimate BitMap file.

Cause: A DrawBitmap command has tried to access an unrecognized image format.
Solution: Verify the image can be displayed, or convert the image to a known format.

Not enough memory to reallocate string.

Cause: A general purpose error that appears if a system call has failed through lack of memory.
Solution: Try shutting down unnecessary applications. If the problem persists, contact VYSOR Integration Inc Technical Support.

Number cannot be negative in this command.

Cause: You used a negative number in a command that supports positive numbers only, such as UsePen.
Solution: Use only positive integers.

Number is too large or too small.

Cause: You have entered a number greater than 2^{31} or less than -2^{31} .
Solution: Check why you need such large numbers or check the input data.

Numeric underflow.

Cause: A math operation has tried to create a negative number less than -2^{31} .
Solution: Adjust the math code so that very large negatives cannot occur.

Numeric overflow.

Cause: A math operation has resulted in a number that will exceed 2^{31} .
Solution: Adjust the math code so that overflows cannot occur.

RETURN without GOSUB.

Cause: Either a missing GoSub command, or you have jumped into a subroutine in error.
Solution: Review the style of your code. Subroutines should have one entry point only. Multiple exit points (i.e. Return commands) are allowable.

Script file is larger than 4 gigabytes.

Cause: Very unusual error. The script file specified appears to be enormous, or is corrupt.
Solution: Check that the script file is in text only format.

Starting location must be greater than 0.

Cause: A SubStr command requires a location value ≥ 1 .
Solution: Correct your script.

Starting location greater than string length.

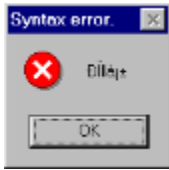
Cause: A SubStr command as tried to access a location outside of the string.
Solution: Correct your script or the code that generates the location.

Syntax error

Cause: Very common fault during development. There is an error somewhere. It may not necessarily be exactly in the line indicated if the fault is unbalanced '}' or ')' characters.

Can also be caused if the script has been saved as other than TEXT only. You will get a get a MessageBox

window with garbage like in the window below, when the program starts.



Typical error window for PiXCL script saved as a .WRI, .DOC or .RTF file.

Solution: Check the line syntax in the indicated line. Re-save the script as a Text Only file. WordPad or NotePad are the ideal editors for scripts.

System shutdown failed.

Cause: Your NT system would not allow the command to be executed, probably because the required privileges were not set.

Solution: Try using the RunExt() command with the Shutdown command in the new script.

Timer not available. Close some applications and restart.

Cause: The system resources required cannot be allocated to PiXCL.

Solution: Close some applications and try again.

Too many GOSUB nesting levels.

Cause: More than 50 nested levels have been defined.

Solution: Reduce the complexity of the code.

Unable to create mutex.

Cause: A SendKeys command using the PiXCLmsg.DLL has failed.

Solution: Check your system resources. A reboot may be required to clear the problem.

Unable to load bitmap file.

Cause: A DrawBitmap command has tried to access an unrecognized image format.

Solution: Verify the image can be displayed, or convert the image to a known format.

Unable to open bitmap file.

Cause: A DrawBitmap command has tried to access an unrecognized image format.

Solution: Verify the image can be displayed, or convert the image to a known format.

Unable to read the specified .INI file.

Cause: The INI file specified either does not exist, does not exist in the directory, or is corrupted in some way.

Solution: Check your PATH. Check the format of the INI file.

Unable to return selected Font name.

Cause: A UseFont command has failed because the Font is unknown or not installed.

Solution: Correct the script, or install the required font.

Unbalanced comment markers

Cause: Very common development error. '{ ' with no '}''.

Solution: Check that the comment markers are balanced.

Internet Technical Support

VYSOR Integration has set up technical support to provide answers to the most frequently asked questions as well as news on products, upgrades and techniques.

There are a variety of files available via anonymous ftp from

<ftp.vysor.com/outgoing>

Our email address is

techsupport@vysor.com

There are also World Wide Web pages at

<http://www.vysor.com>

Registered users have UserID and Password controlled access to their own area where upgrades, bug fixes and development news are provided.

Write to us at

VYSOR Integration Inc.,
91 rue Bocage, Suite B,
Gatineau, Quebec,
Canada J8T 5W5

Attention: Technical Support

or phone Canada (819) 246-7792
fax Canada (819) 568-6859

Software License and Limited Warranty

PiXCL Tools is copyright © (1994-2001) VYSOR Integration Inc. All Rights Reserved.



Portions of the command reference for the IDR series commands that interface to the Idrisi GIS are adapted with permission from copyrighted Idrisi API documents provided by Clark University, Massachusetts.

Attention:

This licensed software is protected by Canadian and International Copyright Law. Read the following Software License Agreement before continuing to use this product. By using this software you signify that you have read this Software License Agreement and accept its terms.

IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT USE THIS SOFTWARE, AND RETURN IT PROMPTLY TO VYSOR INTEGRATION INC.

Software License Agreement

This is an agreement between you and VYSOR Integration Inc ("VYSOR"). By using this software, you are agreeing to become bound by the terms of this agreement.

1. VYSOR Integration Inc grants you a non-exclusive license to use the software on a single computer or cpu. Multiple licenses are available at additional cost.
2. VYSOR retains the copyright, title and ownership of the software and written materials regardless of the form and media of the original. You may make one backup copy for backup purposes. This backup copy must include all copyright notices that appear on the original disks.
3. You may physically transfer the software from one computer to another provided the software is used on one computer at a time. You may not distribute copies of the software or accompanying documentation to others. You may not transfer the software or documentation to any person without the prior written consent of VYSOR. In no event may you transfer, assign, rent, lease or otherwise dispose of the software on a temporary basis.
4. You are not permitted to patch, disassemble or recompile the Windows resources. You are not permitted to add additional resources or modify existing resources. The sole specific exception is the replacement of any of the existing icons for program development purposes, and this must be done using a suitable icon replacement tool, not a resource compiler, as the former does not change the binary file size.
5. This License is effective until terminated. This license will automatically terminate without notice from VYSOR if you fail to comply with the provisions of the License.
6. **DISCLAIMER OF WARRANTIES:** VYSOR disclaims all other Warranties, expressed or implied, including, but not limited to, any implied Warranty of Merchantability or fitness for a particular purpose.
7. **VYSOR EXPRESSLY WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES OF ANY KIND (INCLUDING DAMAGES ARISING FROM ANY THEORY OF LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) WHATSOEVER ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT EVEN IF VYSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**
8. In any event where warranties are found to exist, such warranties shall be limited in duration to thirty (30) days following the date of delivery to you. In no event shall VYSOR's liability to you exceed the amount paid for the license to use the software.
9. This License shall be governed and construed in accordance with the laws of the Province of Quebec, CANADA and shall benefit VYSOR, its heirs, successors and assigns.

OverView

PIXCL and its sibling, geoPIXCL, are 32 bit interpreted graphics, image display and image processing languages for Windows 95/98/ME/NT4/2000. The PIXCL commands allows you to simply perform such diverse functions as building custom menus and customized imaging utilities, including floating toolbars to launch other applications, moving and resizing application windows, and drawing a wide variety of graphic objects using different shapes, colors, and patterns in a window. PIXCL supports 15 of the most commonly used bitmap formats. Here are some examples of what you can do with the PIXCL and geoPIXCL languages:

- Create custom image acquisition, processing, display and printing programs.
- Import and process images from TWAIN-compliant devices such as **scanners** and **digital cameras** from Kodak, Sony, Panasonic, Olympus and many other manufacturers.
- Display, process and convert images from one format to another.
- Develop a Photo-CD image viewer and processor.
- Create front-end control or “glue” programs that link various other applications into a functional suite.
- Write powerful draw programs that accept keyboard and mouse input.
- Build a computer-based interactive training system with images, sound files and On-line Help.
- Create stand-alone CD-ROM titles.
- Build Windows demonstration, marketing and multimedia programs and prototypes complete with custom windows, menus, toolbars, dialogs and messageboxes.
- Create your own install programs that build directories, copy files, and create program groups in Explorer.
- Create your own simple backup utility.
- Customize the layout of your application windows.

Memory Requirements

PiXCL runs in the 32 bit Windows 95 / 98 / ME / NT4 / 2000 environment, and must co-exist with all the other programs, including the Windows video device driver. Here's a summary of the impact PiXCL has on your PC.

- Size of PiXCL50.EXE (around 480KB) plus PiXCLmsg.DLL (18KB)
- Size of PXLimage.DLL (523KB).
- Size of PXLbtmps.DLL (304KB).
- Size of PXLtwain.DLL (92KB).
- Maximum size of a PiXCL script file (1 GB max, but 20 - 100 KB is typical.)
- Amount of dynamically assigned string variable memory (arbitrary).
- Any disk space or memory needed by bitmap files. This can be quite large, but Windows takes care of most of the memory management, and PiXCL takes care most of the rest automatically. There are PiXCL commands for memory management issues related to bitmap images and dialog boxes.
- A PiXCL application does not consume much of the system resources. According the Windows 95 resource meter, when PiXCL is running it uses 2-3 % of system, 1-2 % of user and 10-12 % of GDI resources. By way of comparison, MS-Word 97 requires 6-10% of both system and user, and 10-12 % of GDI resources.

PiXCL is fully compatible with Windows 95/98/ME/NT4 and Windows 2000. The minimum system cpu and memory for any NT system is a Pentium, Pentium II or III, 64 MB, and for Windows 95/98, you should consider 32 MB as the workable absolute minimum, with 64 MB a more realistic number. These are the numbers that the industry suggests for using these operating systems, regardless of the applications running. If you need to run large memory hungry applications such as desktop publishing or graphics manipulation, more memory is always better. For both Windows 95/98 and NT 4.0, a useful and usable system should have at least 64 MB. Windows 2000 really needs 128MB or more to function well. A 2MB vram video card capable of 1024x768x64K colors or better is recommended if you intend to work with images, as this provides a much better color representation than any 256 color mode.

Rules and Syntax

The basic rules and syntax for PiXCL are quite simple. There are five classes of identifiers (i.e. names) in PiXCL: **commands**, **variables**, **tokens**, **labels** and **constants**. Identifiers are not case sensitive. The sections that follow describe additional rules for these identifiers.

Commands

Commands are the basic building blocks of a PiXCL script. Most PiXCL commands follow the syntax

```
DrawText(10,10,"This is a PiXCL program")
```

where a keyword (`DrawText`) is followed by parameters enclosed in parentheses. The keyword names the action the command is to perform and can occur anywhere on a line. Parameters (also called arguments) provide the information necessary to execute the command and are separated by commas. Parameters can be integers, text enclosed in double quotes, variables (either string or integer), and tokens. Commands fall into two categories:

- fixed argument number commands (such as `DrawText` above); and
- variable argument number commands (such as `SetMouse` described later)

Other commands, such as the unstructured `If`, follow the looser syntax

```
If <condition> Then <commands>
```

With these commands, the only rule is that the command elements must be separated by at least one space. You'll find the syntax for all PiXCL commands in the [Command Reference Section](#) of this Help file.

Variables

PiXCL lets you create string, 32 bit integer and 32 bit single precision floating point variables. To create either one, all you have to do is use its name. In **PiXCL 5**, there is support for 64 bit integers and 64 bit double precision floating point, mostly related to getting huge file sizes and passing these values to certain commands. You can use 64 bit variables in **If**, **For** and **While** structures.

Variable names can be any length and can be upper or lower case. Variable names are **case insensitive**. A variable name can use any of the characters **A-Z**, **a-z**, **0-9**, or **_** (underscore), but it cannot start with a number. If a variable starts with a number, it will be flagged as a syntax error when the program is tested or run.

Variables cannot start with, and cannot include the **@** character: this is reserved as the first character in predefined constants, discussed below.

Variables cannot start with, and cannot include the **%** character: this is reserved as the modulus operator.

Variables should not include **[** and **]** characters, as these are required for array variable support.

It is recommended that variables do not include the strings **"Else"** or **"Endif"**, as these are keywords used with the structured **If**. Using these strings in variable names will usually result in a syntax error.

To differentiate a string variable from an integer variable, you must place a dollar sign (**\$**) at the end of the string variable's name, and for real variables, place an ampersand (**&**) at the end of the variable name. For example, PiXCL will treat **Number** as an integer variable, **Number\$** as a string variable, and **Number&** as a real variable.

In **PiXCL 5**, there is also the 64 bit integer **Number#** and double **Number#&**.

Here are some other valid variable names:

```
Mousex      _012      Foxtrot$    y1      NEXT_LINE0  Degrees&
```

When you create an integer or real variable, PiXCL automatically initializes it to zero or 0.0 respectively. Likewise, a string variable is automatically initialized to an empty string (""). You can also initialize variables yourself in the following manner:

```
Counter = 6
FileName$ = "CONFIG."
RealNumber& = 124.4558      or
RealNumber& = 1.244558e002
```

Here `Counter` is set to 6 and `FileName$` is set to the string "CONFIG."

You can optionally use the `Set` command to initialize a variable, as in the following:

```
Set Counter = 6
```

In some situations, you won't need to initialize a variable before using it. For example, some commands will set a variable for you, as in the following command, which lets you get mouse input:

```
SetMouse(1,1,10,10,Mouse_hit,x,y)
```

Here the `x` and `y` variables are set automatically by PiXCL when the command is executed; they indicate the point in the window where the mouse pointer was sitting when the user clicked the mouse button.

You can also perform simple mathematical calculations using integers and store the result in an integer variable. For example, the following command sets the integer variable `Mouse_x2` to the value in `Mouse_x1` multiplied by three:

```
Mouse_x2 = Mouse_x1 * 3
```

Integer variables are 32 bit positive or negative numbers i.e. -2,147,483,647 to 2,147,483,648.

Floating point variables are 32 bit floating point numbers.

In **PiXCL 5**, 32 and 64 bit integer, string, floating point and double arrays are supported. Anywhere you use a variable name, and array variable name of the same type can be used. E.g.

```
Number[0] is an integer array element
Number&[0] is a floating point array element
Number$[0] is a string array element.
Number#[0] is a 64 bit integer array element
Number#&[0] is a double array element
```

The math operators available in PiXCL are

| | |
|---|----------------------------------|
| - | Subtraction |
| + | Addition |
| * | Multiplication |
| / | Division |
| % | Modulus (integer variables only) |

PiXCL also supports the following short forms of post increment and decrement.

| | |
|----|---------------------|
| ++ | Increment by 1 |
| -- | Decrement by 1 |
| += | Increment by number |
| -= | Decrement by number |

For example,

`Counter = Counter + 1` can be written as `Counter ++` or `Counter++` and

`Counter = Counter - 1` can be written as `Counter --` or `Counter--`

There does not need to be a space after the variable name. The short form is useful in processing loops, and will execute slightly faster than the longer format.

Examples of incrementing or decrementing by a number are

`Counter = Counter + 6` can be written as `Counter += 6` or `Counter+=6` and

`Counter = Counter - 4` can be written as `Counter -= 4` or `Counter-=4`

PiXCL also provides a positive integer random number generator command, [Random\(Range,Number\)](#).

In the case of string variables, you can perform concatenation. For example, the following command appends the string "Consultants" to end of "Digital " and places the result in the string variable `CompanyName$`:

```
CompanyName$ = "Digital " + "Consultants"
```

See the [Set](#) command in the next Section for more information on variables.

Note: PiXCL lets you re-use string variable memory by freeing variables that you no longer need; see the [FreeVar and FreeVarAll](#) commands in the next Section.

Tokens

Many PiXCL commands require that you use tokens as parameters. A token is a special identifier that has been predefined by PiXCL. For example, in the following command syntax, **PIXEL** and **METRIC** are tokens:

```
UseCoordinates (PIXEL/METRIC)
```

For this command, you must use either PIXEL or METRIC for the parameter, and you must spell the token correctly. No other parameter will be accepted, and will cause a Syntax Error.

When a command requires a token, the token appears in upper case in the command syntax, although the tokens are not case sensitive. You'll find as you create your PiXCL programs that it's a good idea to follow this same convention.

Labels

Labels follow the same naming conventions as variables in PiXCL. For example, they can be any length and can be upper or lower case. Labels have the additional restrictions that they must be placed at the **start of a line** (in the first column of the line), and they must end with a **:** (colon). For example, here are some valid and invalid labels:

```
Next:           {A valid label}
```

```
    Wait_for_input:
```

```
    {An invalid label because it isn't located at the start of the line}
```

It is recommended that labels do not include the strings "**Else**" or "**Endif**", as these are keywords used with the structured **if**.

Constants

There is a small number of defined constants that can be used within a PiXCL script, and these are preceded by an ampersand **@** character.

PiXCL provides seven predefined integer constants for the Windows Registry access to permanently open keys.

@RDB_CLASSES_ROOT
@RDB_CURRENT_USER
@RDB_LOCAL_MACHINE
@RDB_USERS
@RDB_CURRENT_CONFIG
@RDB_PERFORMANCE_DATA
@RDB_DYN_DATA

A constant can be used in place of any integer argument where the substitution makes sense. For example, to see the actual values of the above constants, the following commands can be used.

```
DrawText(10,10,"Constant Substitution")
DrawNumber(10,35,@RDB_CLASSES_ROOT)
DrawNumber(10,60,@RDB_CURRENT_USER)
DrawNumber(10,85,@RDB_LOCAL_MACHINE )
DrawNumber(10,110,@RDB_USERS)
DrawNumber(10,135,@RDB_PERFORMANCE_DATA)
DrawNumber(10,160,@RDB_CURRENT_CONFIG )
DrawNumber(10,185,@RDB_DYN_DATA )
```

PiXCL also provides these logical (integer) constants as well.

@TRUE **set to 1**
@FALSE **set to 0**
@YES **set to 1**
@NO **set to 0**

PiXCL also provides these real constants as well.

@AMC (Atomic Mass Constant) **set to 1.66043E-27**
@AVOGADRO (Avogadro's Number) **set to 6.02252E23**
@BOLTZMANN (Boltzman's Constant) **set to 1.38054E-23**
@DEG2RAD (Degree to Radians) **set to 0.017453292519943**
@E (Naperian log) **set to 2.718281828459045**
@ELECTRIC (Electric Field Constant) **set to 8.8541853E-12**
@EULERS (Euler's Constant) **set to 0.5772156649015388**
@FARADAY (Faraday Constant) **set to 9.64870E4**
@GFTSEC (Gravitation Acceleration, ft/sec) **set to 32.174**
@GMTSEC (Gravitation Acceleration, m/sec) **set to 9.80665**
@GRAVITATION (Gravity Constant) **set to 6.670E-11**
@LIGHTVEL (Light speed in m/sec) **set to 2.997925E8**
@MAGFIELD (Magnetic Field Constant) **set to 1.256637**
@PI **set to 3.141592653589793**
@RAD2DEG (Radians to Degrees) **set to 57.29577951308232**

Comments

All characters between { and } are treated as comments by PiXCL. You can place comments anywhere in a PiXCL text file. You can also safely nest comments.

For example, the following program draws the cars bitmap (CARS.BMP) located in the Windows directory in a continuous line across the screen. The program is generously commented to make it easier to read. This program draws the cars bitmap across the screen a set number of pixels apart.

```
{-----CARS.PXL-----}
{Initialize}
UseCoordinates(PIXEL)    {Use pixels, not millimeters}
                        {Starting x coordinate}
                        {Starting y coordinate}
Step = 32                {Step by 32 pixels at a time}

{Get the screen's width in pixels}
GetScreenCaps(HORZRES, Pixels)
{Maximize the window}
SetWindow(MAXIMIZE)
{Get the Windows directory and build the path to CARS.BMP}
DirGetWindows(WindowsDir$)
CarsPath$ = WindowsDir$ + "\CARS.BMP"
{Loop to draw the cars bitmap across the screen}
Next: DrawBitmap(x,y,CarsPath$)
x = x + Step
If x < Pixels Then Goto Next

{Leave the finished window up until the user kills it}
WaitInput()
```

White Space

White space is a general term for the elements that PiXCL ignores in a script. PiXCL treats as white space all blanks, tab characters, carriage returns, line feed characters, split vertical bars (|), and comments. White space is ignored at any point in a script.

Starting a Program

To start a PiXCL program from the command prompt, you must provide the name of the PiXCL executable file (PiXCL50.EXE) followed by the name of the script file. For example, if PiXCL is located in the C:\PiXCLTools directory and your script file is located in C:\WORK and is named SCRIPT.PXL, you would use the following command line:

```
C:\PiXCLTools\PiXCL50 C:\WORK\SCRIPT.PXL
```

If you start PiXCL without providing a script file name, you'll see a dialog box that prompts you for a script file.

Note: Because PiXCL's command-line syntax is similar to Notepad's (or any other application that lets you load a file on startup), there are several ways you can simplify it. For example, if you've created a file association linking .PXL files to PiXCL50.EXE, all you need to provide on the command line is the name of the PiXCL script file, as in SCRIPT.PXL.

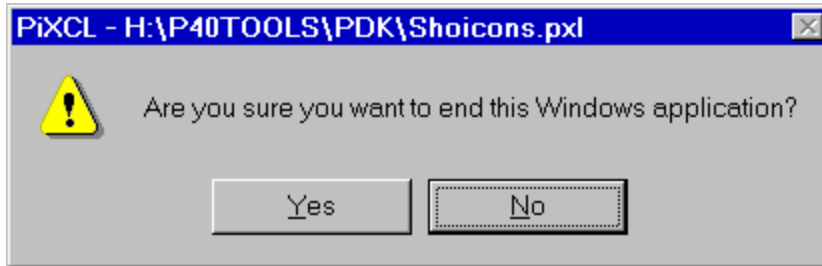
In PiXCL 4.0 and later, the initial Window that is created is hidden. Your script should call either or both [WinLocate\(...\)](#) or [WinShow\(...\)](#) (described further below) as the first commands to define the window starting location, styles and extended styles.

PiXCL also supports an arbitrary number of additional command line arguments which can be accessed from within the PiXCL application script.

See also [Building RunTime EXE](#) files.

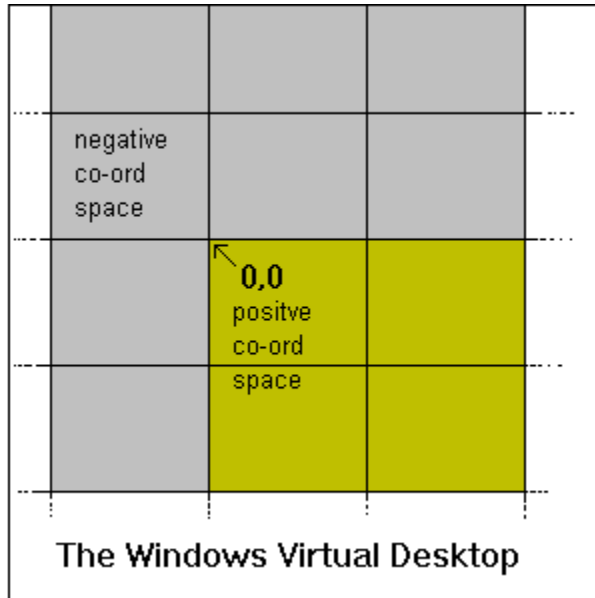
Stopping a Program

To stop a PiXCL program at any point, press CTRL+BREAK. PiXCL will display the message box shown below. To end the program, select Yes. To have the program resume where it left off, select No.



The Coordinate System

In PiXCL the upper-left corner of your desktop window is the origin, or point (0,0). The X coordinate increments to the right along the positive **x** axis, and the Y coordinate increments towards the bottom along the positive **y** axis. The actual window coordinate space is much larger, and extends into the negative X and Y coordinate space. This larger space is often referred to as the virtual desktop. Graphically, you can see how this appears in the image below.



For Windows 98 and 2000, multiple monitors are supported when additional video cards are installed in the system. The effect is that each monitor provides a window into the virtual desktop. Please see the Windows 98 / ME / 2000 help on setting up multiple monitors for more information.

See also the [UseCoordinate\(PIXEL | METRIC\)](#) command, and the [WinLocate](#) and [WinGetLocation](#) commands that either accept or generate positive and negative position arguments.

Drawing Tools

PiXCL provides you with a wide variety of tools to draw within its window client area. It lets you set up pens to draw lines, rectangles, ellipses and polygonal shapes, brushes and user defined patterns to fill interiors and exteriors, and fonts to write text. To create tools for drawing geometrical shapes, you use commands such as [UsePen](#) and [UseBrush](#). To create fonts for writing text, you use the [UseFont](#) command.

Note#1: If you do not establish any drawing tools before drawing in a window, PiXCL uses these default drawing tools: a black pen, a white brush, and the System font.

Note#2: It is not possible to draw in the menu bar or title bar, or in any other Window not created by PiXCL.

Setting and using Background Colors

The default PiXCL application client area color is light gray (R,G,B = 192,192,192). You can set the desired background color with the [UseBackground](#) command followed by a [DrawBackground](#) command at any time. If necessary, you can retrieve the current background colors with the [GetBackground](#) command.

Using Pens

The [UsePen](#) command lets you assign a pen for drawing lines and borders. Pens can be solid, dashed, dotted, and more. For example, the following example creates a solid black pen, two pixels wide:

```
Width=2
```

```
UsePen(SOLID,Width,0,0,0)
```

The Width argument controls the width of the pen in pixels or millimeters (depending on the co-ordinate system in use). The three arguments following the Width argument specify the color of the pen. They control the intensity of the colors red, green, and blue in that order. In the example above, all the colors have 0 intensity, so the pen will be black. Conversely, the following line would create a white pen:

```
UsePen(SOLID,Width,255,255,255)
```

The pen you specify with UsePen will be used in all subsequent drawing operations, or until you use UsePen again to change the pen.

Note: The default pen is solid, black, and has a width of 1 pixel. If you use a command that draws a shape, but you haven't yet set up a pen with the UsePen command, PiXCL uses the default pen.

Using Brushes

The UseBrush command lets you establish brushes for drawing and filling areas in rectangles, ellipses, pies, and the like. You can create brushes that are solid or hatched, have diagonal lines, horizontal lines, vertical lines, and more. For example, here's the command to create a solid blue brush:

```
UseBrush(SOLID,0,255,0)
```

As you might have guessed, the last three arguments control the color of the brush.

Note: The default brush is solid white.

You can also create your own brush patterns (8x8 in Windows 95/98, up to 256x256 in Windows NT) and use these with the flood and fill commands. For example,

```
LoadBitmap(Pattern1$,FULL)
```

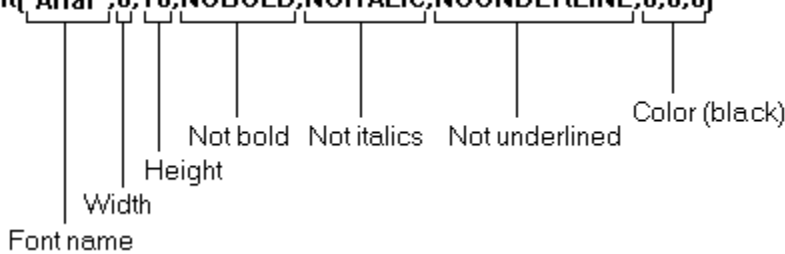
```
UseBrushPattern(Pattern1$)
```

loads a user defined bitmap pattern and sets the current brush.

Using Fonts

You establish a font in PiXCL by using the [UseFont](#) or [UseFontExt](#) command and giving it a series of font attributes, including the font name, width, height, style (bold, italics, or underline), and the color. PiXCL uses the font you've established the next time you draw text or numbers on the screen. Here is an example of the [UseFont](#) command.

```
UseFont("Arial",6,10,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
```



Font name

Width

Height

Not bold

Not italics

Not underlined

Color (black)

Note#1: PiXCL supports all the fonts installed in your system, including TrueType fonts and fonts installed with third-party font packages, such as the Adobe Type Manager.

Note#2: By default, PiXCL uses a black System font.

Note#3: The [Button](#), [ComboBox](#) and [SetEditControl](#) commands draw text in the current font.

Note#4: PiXCL has a default font escapement angle of 0, for horizontal text. This can be changed with the [SetFontEscapement](#) command. The current escapement value is obtained with the [GetFontEscapement](#) command.

Note#5: You can load and unload any font that exists on your disk, with the [AddFont](#) and [RemoveFont](#) commands.

Drawing and Writing

PIXCL provides a variety of commands for drawing and writing output to the screen. The following sections give a brief overview of these commands. For more detailed descriptions of these commands, see the command reference later in this chapter..

Drawing Text

To draw text in a window, you use the [DrawText](#), [DrawShadowText](#), [DrawTextExt](#) or the [DrawShadowTextExt](#) command. For example, the following command displays the text "The ABC Company Shell" using the current font. The starting position of the text is at the point (20,10).

```
DrawText(20,10,"The ABC Company Shell")
```

You can also specify a rectangle in which text is drawn in the current font, with automatic word-wrap and definable justification, using the [DrawTextExt](#) command. This command draws the string centered in the rectangle specified.

```
DrawTextExt(20,10,120,45,"The ABC Company Shell",CENTER)
```

If you want to display a number in a window, you can use the [DrawNumber](#) command. For example, the following command displays the number 200 starting at the point (50,60):

```
DrawNumber(50,60,200)
```

[DrawText](#), [DrawTextExt](#), [DrawShadowText](#), [DrawShadowTextExt](#), and [DrawNumber](#) are also convenient for displaying the contents of variables in a window.

You can also set the draw angle and character spacing with the [SetFontEscapement](#) and [SetTextSpacing](#) commands, as well as add and remove any font from the Windows font table with the [AddFont](#) and [RemoveFont](#) commands.

Drawing Lines and Shapes

PIXCL provides a range of commands for drawing lines and shapes, as shown in a partial list Table 1. All of these commands use the current pen to draw borders and the current brush to fill interiors.

| Command | Purpose |
|------------------------------------|--|
| DrawArc | Draws an arc. |
| DrawChord | Draws a chord. |
| DrawEllipse | Draws an ellipse (or circle). |
| DrawFlood | Floods an area with color using the current brush. |
| DrawIcon | Draws one of the PIXCL or system icons. |
| DrawLine | Draws a line. |
| DrawPie | Draws a pie wedge. |
| DrawPolygon | Draws a polygon with up to 32 vertices. |
| DrawRectangle | Draws a rectangle. |
| DrawTriangle | Draws a triangle. |
| DrawEdgeRectangle | Draws a rectangle edge in several styles. |
| DrawRoundRectangle | Draws a rectangle with rounded edges. |

[DrawShadeRectangle](#) Draws a color gradient rectangle.

Table 1: Commonly used commands for Drawing Lines and Shapes

The following example shows how to use the [DrawLine](#) command to draw a line from the point (10,80) to the point (100,20):

```
DrawLine(10, 80, 100, 20)
```

To draw a rectangle, you use the [DrawRectangle](#) command. The following command draws a rectangle that has its upper-left and lower-right corners at the points (15,25) and (75,110):

```
DrawRectangle(15, 25, 75, 110)
```

This function uses the current pen to draw the border of the rectangle and the current brush to fill the interior.

The [DrawEllipse](#) command lets you draw a circle or an ellipse. The following example draws an ellipse that is bounded by the rectangle specified by the points (100,30) and (250,90):

```
DrawEllipse(100, 30, 250, 90)
```

As with the [DrawRectangle](#) command, the [DrawEllipse](#) command uses the current pen to draw the border of the rectangle and the current brush to fill the interior.

Drawing and Using Bitmaps

PiXCL has commands for placing the contents of Windows bitmap (BMP, JIF, JPEG, PCD, PCX, PPM, PNG, PSD, RAS, RLE, TGA, TIF) files on the screen: [DrawBitmap](#), [DrawSizedBitmap](#), [DrawTrBitmap](#), [DrawTrSizedBitmap](#) and [DrawZoomedBitmap](#). The [Draw\[Tr\]Bitmap](#) commands are the simpler of these: It lets you locate a bitmap starting at a specified point in a window. For example, the following program places the Winlogo bitmap (WINLOGO.BMP) starting at the point (10,10):

```
Winlogo$ = "C:\WINDOWS\WINLOGO.BMP"  
DrawBitmap(10,10,Winlogo$)  
WaitInput()
```

The [Draw\[Tr\]SizedBitmap](#) commands lets you stretch or compress a bitmap to fit within a specified rectangle. You indicate the upper-left corner of the rectangle using the first two parameters and the lower-right corner using the second two. For example, the following program displays the Winlogo bitmap within the condensed rectangle specified by the points (10,10) and (100,120):

```
Winlogo$="C:\WINDOWS\WINLOGO.BMP"  
DrawSizedBitmap(10,10,100,120,Winlogo$)  
WaitInput()
```

You can also use the [DrawSizedBitmap](#) command to invert a bitmap as you place it on the screen (see the [DrawSizedBitmap](#) command in the command reference for more details).

[DrawSizedBitmap](#) can also be used to load a bitmap without displaying it, by setting all the coordinates to zero. There is also the equivalent [LoadBitmap](#) command. These two commands are equivalent.

```
DrawSizedBitmap(0,0,0,0,ImageFile$)  
LoadBitmap(ImageFile$,FULL)
```

There is also a potential system memory limit when drawing bitmaps, and especially sized bitmaps. If you want to display an especially large image (i.e. more than about 3 MB), Windows may be unable to assign enough memory for the operation, or may bog down in swapping data to and from the disk.

PIXCL also has a command, [GetBitmapDim](#) that accesses the supported format bitmaps, and returns the number of lines, pixels and bits per pixel. This can be very useful in deciding whether to use [DrawBitmap](#) or [DrawSizedBitmap](#), and where to draw the bitmap in your application client area.

Other bitmap processing commands include [DrawZoomedBitmap](#) for zoom and roam operations, and all the image processing commands. The [DrawZoomedBitmap](#) can also be used to animate images that are segmented into frames.

When you need to transparently overlay bitmaps, there are the [DrawTrBitmap](#) and [OverlayImage](#) commands that let you specify the transparency color.

PIXCL also supports a comprehensive set of point and geometric [image processing commands](#), including image enhancement and resampling, filtering and and bitmap format conversion. All the writable supported bitmap formats can be saved back to disk.

Flow Control Commands

PiXCL has seven classes of commands to control the flow of programs: unstructured [If-Then](#) and [Goto](#), structured [If-Else-Endif](#), structured [For-Next](#), structured [While-EndWhile](#), structured [Switch-Case-Endswitch](#), plus [Gosub](#) and [Return](#).

The unstructured [If](#) command lets you make a decision when there are two alternative outcomes. It tests the value of a condition, and if that condition is true, the program continues executing commands on the same line following the [Then](#). If the condition is false, the program begins executing commands on the next line following the [If](#).

For example, the following [If](#) command tests the value of the variable Green to see if it is greater than 255. If it is, PiXCL executes the [Goto](#) command on the same line. Otherwise, it executes the [WaitInput\(\)](#) command on the next line

```
If Green > 255 Then Goto Exit
WaitInput()
Exit:
```

The [Goto](#) command transfers control unconditionally to a label. In the previous example, the [Goto](#) command causes the program to branch to the label Exit.

The [Gosub](#) command lets you execute a block of code as a subroutine. When the subroutine is completed, PiXCL executes the next command following the [Gosub](#).

Supplementing PiXCL's flow of control commands

With a little creativity you can easily use [If](#), [Goto](#), and labels to supplement PiXCL's structures for controlling program flow. Here are some examples:

IF / ELSE / ENDIF

```
If <condition> Then <do something> | <do something else> | Goto EndStr
    <do something different>
EndStr:
```

FOR

```
StartLoop:
If <counter> = <value> Then Goto EndLoop
<a command>
<the next command>
<counter> = <counter> + 1
Goto StartLoop
EndLoop:
```

WHILE

```
BeginWhile:
If <counter> = <value> Then Goto EndWhile
<a command>
<the next command>
<counter> = <counter> + <some value>
Goto BeginWhile
EndWhile:
```

Structured If-Else-Endif in PiXCL

In PiXCL, the basic `If <condition> Then <action#1 actions#2 action#n>` statement expects that at least one command or action exists on the current line. Each action must be delimited by a suitable whitespace character ("|" is often suitable). The `If` construction is terminated by a newline character.

In PiXCL, the structured command `If...Else...Endif` is supported according to the following rules.

The `If...Endif` structure is

| | |
|-----------------------------------|---|
| <code>If <condition></code> | If the condition is false, script interpretation jumps to the next command following the <code>Endif</code> keyword. |
| <code> action#t1</code> | |
| <code> action#t2</code> | |
| <code> ...</code> | |
| <code> action#tn</code> | Any whitespace characters after the <code><condition></code> and before the newline character are ignored, as usual. |
| <code>Endif</code> | All additional commands are interpreted until the <code>Endif</code> keyword is located. |

and, extending the command with an `Else` structure ...

| | |
|-----------------------------------|---|
| <code>If <condition></code> | If the condition is true, script interpretation jumps to the next command following the <code><condition></code> . Once the <code>Else</code> keyword has been located, execution jumps to the next instruction after the <code>Endif</code> keyword. |
| <code> action#t1</code> | |
| <code> action#t2</code> | |
| <code> ...</code> | |
| <code> action#tn</code> | |
| <code>Else</code> | Any whitespace characters after the <code><condition></code> and before the newline character are ignored, as usual. |
| <code> action#f1</code> | |
| <code> action#f2</code> | |
| <code> ...</code> | |
| <code> action#fn</code> | If the condition is false, script interpretation jumps to the next command following the <code>Else</code> keyword. |
| <code>Endif</code> | All additional commands are interpreted until the <code>Endif</code> keyword is located. |

Embedded `If-Else-Endif` and `If-Then` statements are supported in all versions, up to 16 levels.

Structured For-Next in PiXCL

PiXCL supports `For` loops with the following general syntax.

```
For variable=n|variable To m|variable [By p|variable]
    ... commands
If <condition> Then Break {optional}
    ... commands
Next
```

Structured While-EndWhile in PiXCL

PiXCL supports `While` loops with the following general syntax. The Test variable can be a number or a string, must be initialized beforehand. For example a `While` loop that tests a numeric variable ...

```
Count = 0
While Count <= 5
    ...commands
Count++
```

```
EndWhile
```

and a **While** loop that tests a string variable ...

```
Count$ = "A"  
While Count = "A"  
    ...commands  
    If <condition> Then Count$ = "B"  
    ... commands  
EndWhile
```

PIXCL 5: Structured Switch-Case-EndSwitch

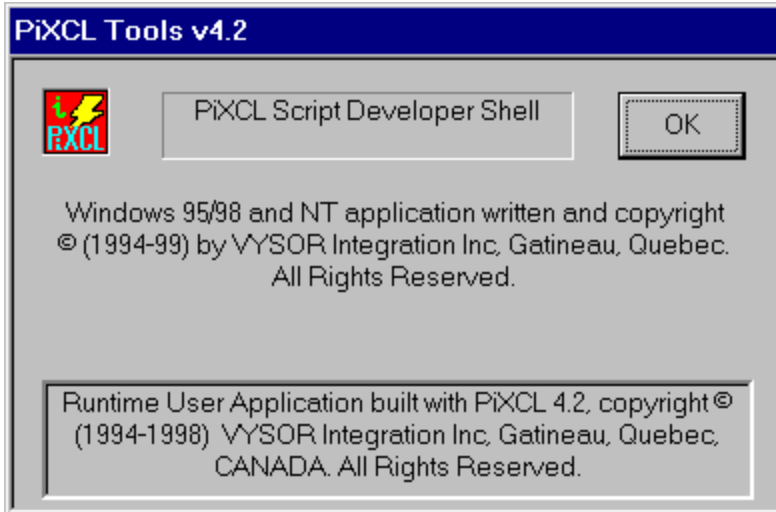
PIXCL 5 supports the useful Switch structure, where you need to process more than two possible values of an integer variable. For example,

```
X = A * B {where A, B have varying values}  
Switch (X)  
Case (1)  
    ...commands  
    Break  
Case (2)  
    ...commands  
    Break  
Default  
    ...commands  
EndSwitch
```

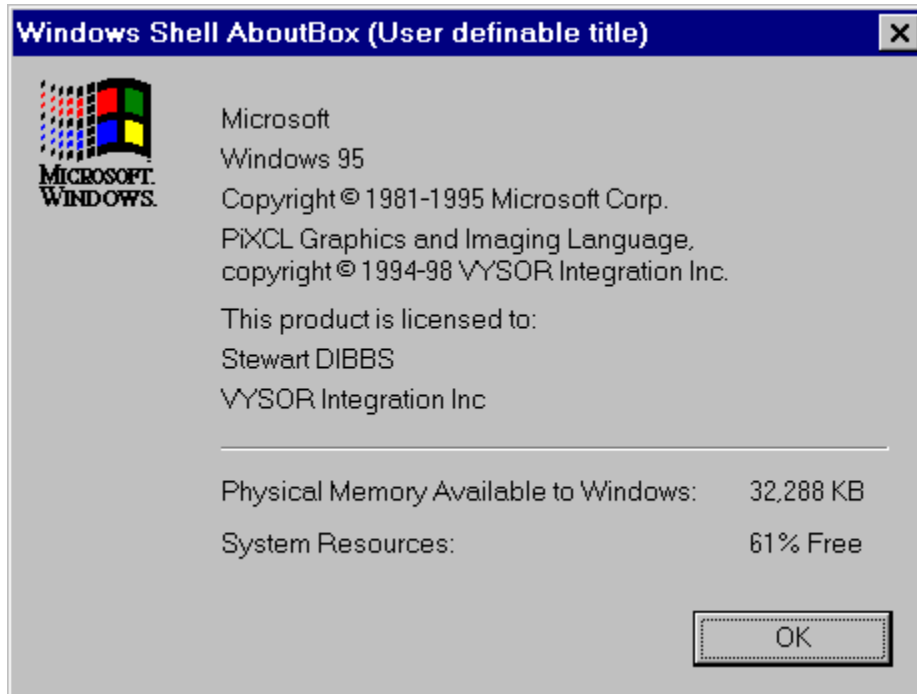
The **Break** command moves execution of the script to the command following the **EndSwitch** keyword. The above can also be done with a series of **If-Endif** commands.

About Boxes

The Windows application style recommendations suggest that an AboutBox that describes the application is desirable. PiXCL provides several options depending on what information you want to provide to the user. There is the [AboutUser](#) box that might appear like the image below. Title and two text regions are user definable.



There is also the [ShellAbout](#) command that displays a dialog that reports some system information, plus some user defined text. An example is shown below.



Message Boxes

The [MessageBox](#) command lets you create your own custom message boxes with one, two or three buttons. For example, the following command creates a message box with OK and Cancel buttons and a question-mark icon. (The second parameter, 1, causes the OK button to be highlighted.) In addition, the message box displays the text "Do you want to exit?" and uses the caption "Exit box."

```
MessageBox(OKCANCEL,1,QUESTION,  
"Do you want to exit?","Exit box",Button)
```

The button you select is returned in the Button variable, starting from the left button numbered 1. The available MessageBox button style TOKENS which also define the button text are

The available Windows MessageBox and built into PiXCL icon style TOKENS are

OK
YESNO
OKCANCEL
RETRYSUCCESS
YESNOCANCEL
ABORTRETRYIGNORE

QUESTION
EXCLAMATION
INFORMATION
STOP
NOICON
ICON01-ICON19

The message text string can be multiple lines if you either

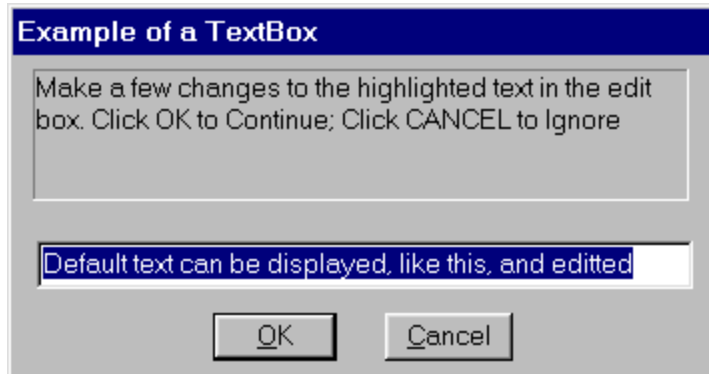
- a) add carriage returns in the message; or
- b) type it all on one line.

Windows will sort out how the message text is displayed.

PiXCL also provides custom dialog boxes with up to sixty pushbuttons, radio buttons, checkboxes, edit controls, comboboxes, list boxes and static text with the [DialogBox](#) command.

Text Boxes

PiXCL's [TextBox](#) and [TextBoxExt](#) commands lets you solicit input from the user. They display a dialog box with a single-line edit control. For example, the following commands produce the text box shown below.



```
Text$ = "Make a few changes to the highlighted text in the edit box. Click OK to continue;  
Click CANCEL to Ignore"  
Caption$ = "Example of a TextBox"  
Score$ = "Default text can be displayed, like this, and edited"  
TextBox(Text$,Caption$,Score$,ButtonPushed)
```

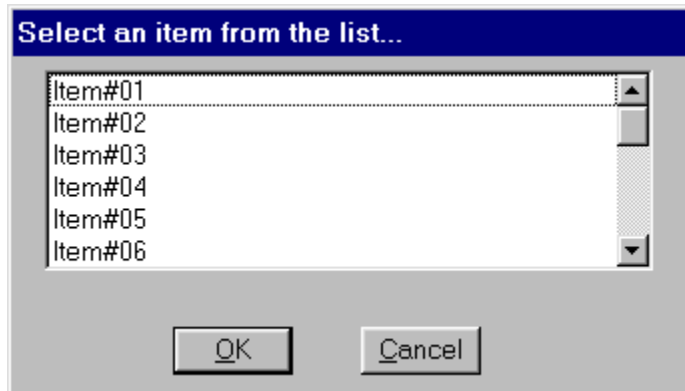
The button you select is returned in the ButtonPushed variable. TextBoxExt is similar, and provides a third Help button that displays a MessageBox with the help string defined in the command.

PiXCL also provides custom dialog boxes with up to sixty pushbuttons, radio buttons, checkboxes, edit controls, comboboxes, list boxes and static text with the [DialogBox](#) command.

List Boxes

The [ListBox](#) and [ListBoxExt](#) commands display a dialog box with a list box inside, so that you can choose from an alphabetically sorted list of items. [ListBoxExt](#) supports multi-column lists and multi-item selections, plus user defined context Help.

For example, the following commands produce the list box shown below.



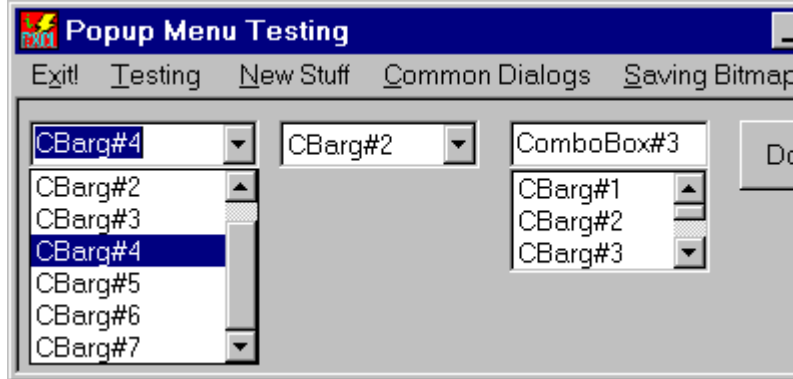
```
Caption$ = "Select and item from the list"  
List$ = "Item#01;Item#02;Item#03;Item#04;Item#05;Item#06"  
Delimiter$ = ";"  
ListBox(Caption$,List$,Delimiter$,Result$)
```

Notice that the list box gets its contents from the List\$ string variable, and that the items in List\$ are separated from one another by semicolons. (PiXCL returns the chosen string in the Result\$ string variable.) Delimiters can be any character you choose. The most commonly useful are semi-colon (";") and pipe ("|").

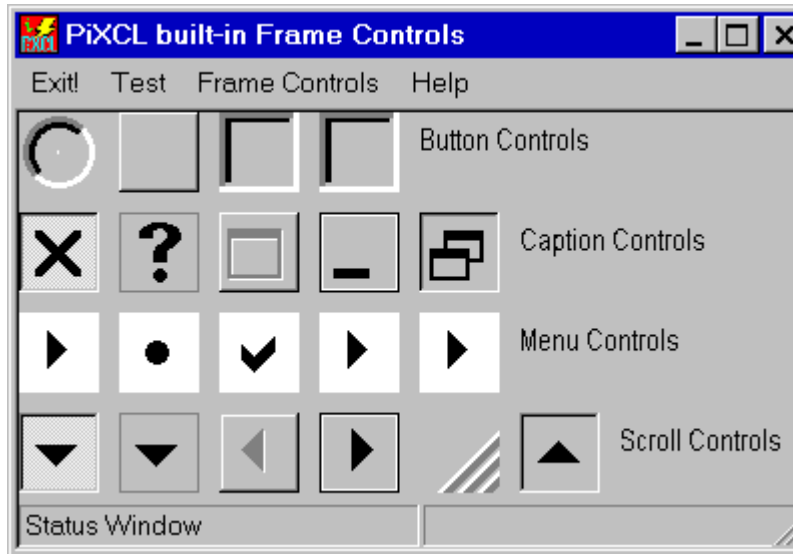
PIXCL also provides custom dialog boxes with up to sixty pushbuttons, radio buttons, checkboxes, edit controls, comboboxes, list boxes and static text with the [DialogBox](#) command.

Combo Boxes

The [ComboBox](#) command produces edit controls with drop down lists, as shown in the figure below. A combo box can include a variety of inputs, such as 3D buttons, radio buttons, and text or numeric input. The [DrawFrameControl](#) , [SetEditControl](#) and [Button](#) commands in PiXCL allow you to create all these styles of client area dialogs.

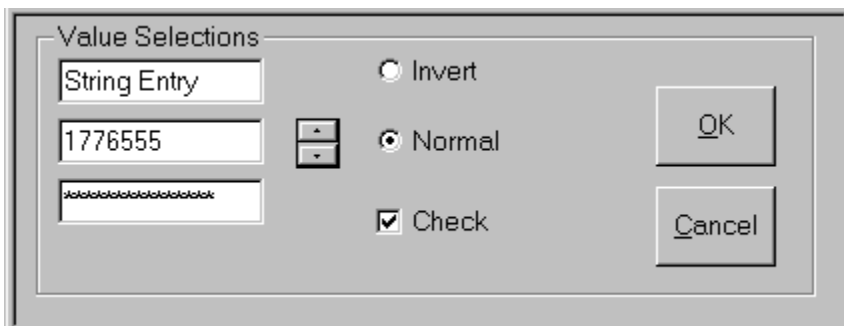


You would use a one of the ComboBox styles where you have multiple lists of items that require the user to select, and a list box dialog is not the best solution.



Frame controls are used to assist in the creation of dialogs that uses the various mouse commands to activate them.

The set of frame controls. Not all styles are shown.



Button styles and Edit controls.

One or more of the following mouse commands are used with a [DrawFrameControl](#) command.

| | |
|-------------------------------------|--|
| SetMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetShftMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetCtrlMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetDbfMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetRightMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetShftRightMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetCtrlRightMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetDbfRightMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetMidMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetShftMidMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetCtrlMidMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| SetDbfMidMouse() | Text or numeric entry edit windows, or radio buttons are activated by clicking in each window. |
| DrawRectangle() | Draw an edit window of a specified size. |
| DrawFlood() | Fill inside a region (e.g. a background area) with a color |
| DrawFloodExt() | Fill outside a region (e.g. a background area) with a color |
| DrawIcon() | Draws one of the PiXCL or system icons at any size and coordinates. |
| DrawEllipse() | Simulate a radio button. The button should be mouse active. |
| SetKeyBoard() | Set up a series of alphabetic or numeric key entry handlers. |
| UseBrush() | Sets an edit window fill color. |
| UsePen() | Sets edit window border color. Black is commonly used. |
| Button() | Draws pushbuttons, radio buttons, checkboxes and group boxes. |
| SetEditControl() | Draws any number of edit controls for string or numeric input. |

See also the PXL code sample program, [KEYBOARD.PXL](#).

ComboBoxes are drawn in the application client area. Note that buttons and edit controls are actually child windows of the PiXCL application. Buttons have the highest display priority, or Z-order, so if the positions of a button and an edit control overlap, the button will appear to overlap the edit control.

PiXCL also provides custom dialog boxes with up to sixty pushbuttons, radio buttons, checkboxes, edit controls, comboboxes, list boxes and static text with the [DialogBox](#) command.

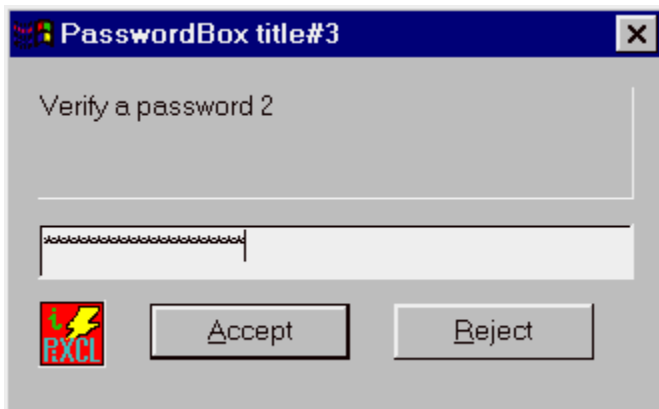
ImageBoxes and PasswordBoxes

PiXCL also supports two additional dialog box styles, the [ImageBox](#) and [PasswordBox](#).



An image box provides a thumbnail image and eleven line text display dialog, with two user defined buttons, and is generally used to provide information about an image or images on your hard disk.

The Password box is similar in appearance to a TextBox, but provides a secure text entry. Any text entered appears as a string of asterisk characters, as shown in the figure below. The title bar, instruction text and the labels on the buttons are programmable.

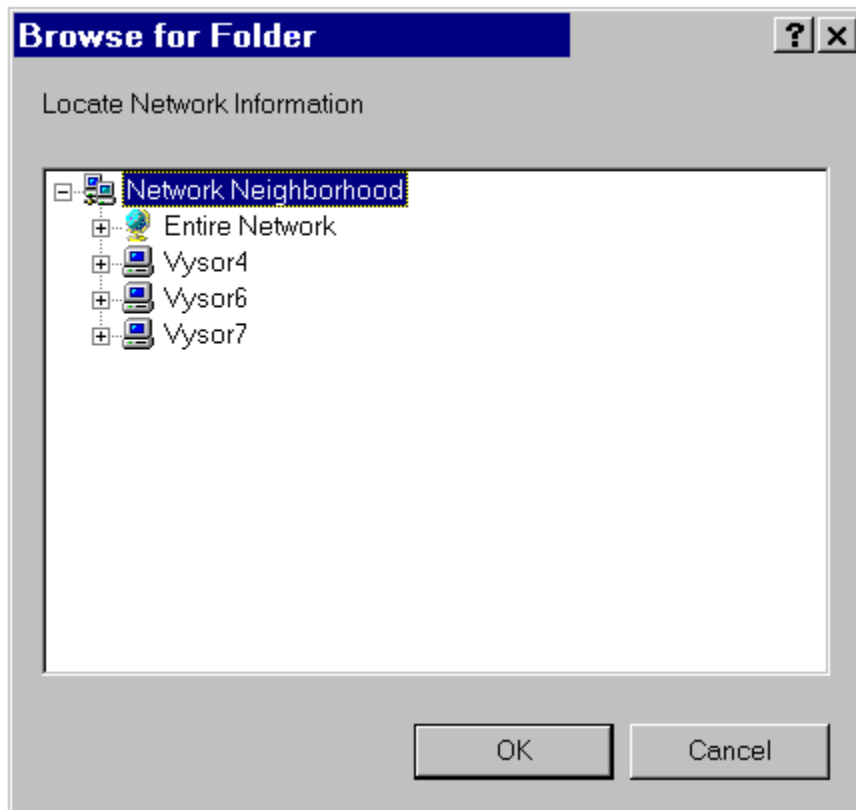


PiXCL also provides custom dialog boxes with up to sixty pushbuttons, radio buttons, checkboxes, edit controls, comboboxes, list boxes and static text with the [DialogBox](#) command.

Windows Shell Dialogs

In PiXCL 4.40 and later, there is access to a number of dialogs that are built into the Windows Shell. The Windows Explorer uses many of these for searching and reporting file information. Full Shell functionality is available with the system file SHELL32.DLL v4.72 or later. If you have Windows 98 or 2000, or Windows 95/NT with Internet Explorer 4.01 or later and with Active Desktop installed, you have the latest shell version already.

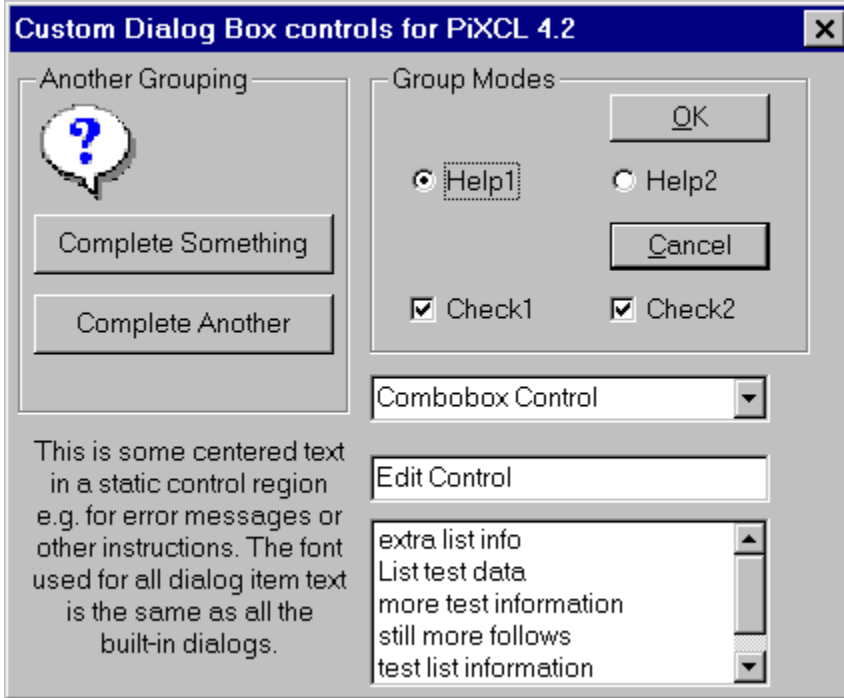
The Shell [GetSpecialFolder](#) command provides a way to get various types of file, system and network information, with a dialog as shown below.



Custom DialogBoxes

In PiXCL 4.20 and later, you can define your own custom popup modal dialog boxes, with combinations of push buttons, radio buttons, checkboxes, group boxes, edit controls, static text, List box, and combo boxes, with the [DialogBox](#) command. These same input controls can be created in the PiXCL client area using other commands.

[DialogBox](#) provides you with the tool to make a exactly the user interface you want, in a separate window, rather than with the [Button](#), [SetEditControl](#) and [ComboBox](#) commands in the PiXCL main client area.

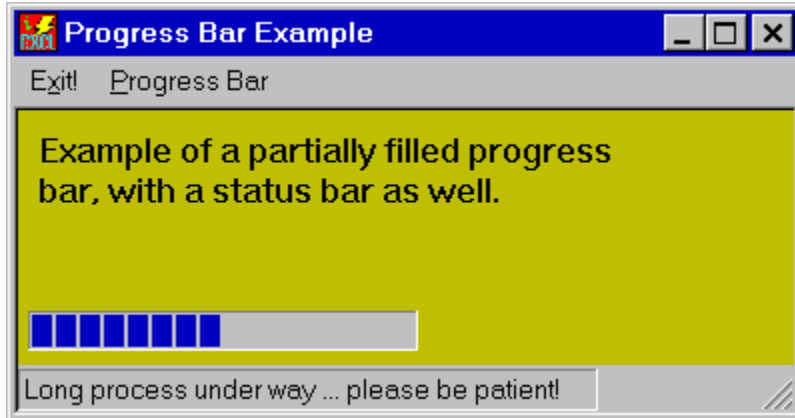


You can include any combination of the dialog items shown above, and position the custom dialog anywhere relative to the PiXCL client area, or centralized in the screen. Up to sixty controls of any of the above types can be created in a custom dialog box.

There are code writing Helper applications with the PiXCL MDI editor that assist in the development of DialogBox code. The **cre8tapp.pxI** sample generates a working application and also includes a selection of ready-made dialog templates that can be pasted into your application. There is also a dialog editor application in which you create the dialog visually, then the template code is pasted into you application.

Windows 95 /98 / NT4 / 2000 Common Controls

Windows 95/98/NT4/2000 include a set of common controls, accessed in a file called COMCTL32.DLL that is found in the system directory. These common controls include multi-part status bars which generally appear at the bottom of the application client area, progress bars that get updated under program control as the application proceeds to completion, trackbars, and a selection of frame control button bitmaps.



The example window, above, shows both a status bar and a progress bar.

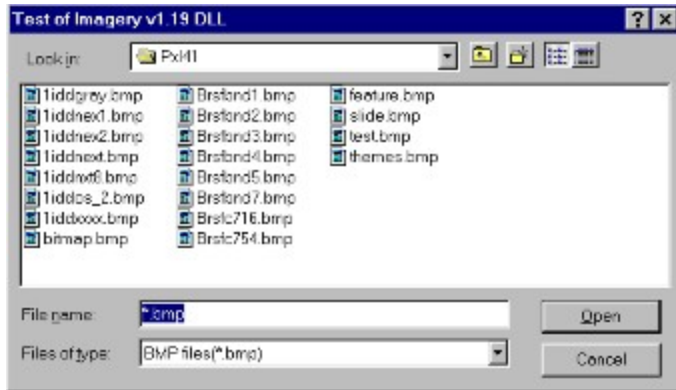
Both types of common control are child windows of the PiXCL application, and must be enabled or disabled under program control.

The commands available to use are

[StatusWindow](#), [DrawStatusWinText](#), [ProgressBar](#) and [UpdateProgressBar](#), [DrawFrameControl](#)

Getting a Filename

PIXCL's [FileGet](#) and [FileSaveAs](#) commands lets you get a filename using the same FileOpen / FileSaveAs common dialog boxes that appears in many Windows applications. An example is shown below. This uses the Windows library COMDLG32.DLL. PIXCL provides commands to access the ChooseFont, ChooseColor and printer common dialogs as well.



In PIXCL 4.10 and later, the FileGet command has been extended to support single and multiple file selection, as well as checking if the selected or entered file and path exists. A command option adds a Help button which displays a MessageBox with user defined title and text.

Managing Windows

PIXCL offers the following commands for working with application windows:

| | |
|----------------------------------|--|
| EnumWindows | Create a delimited list of all parent window title strings. |
| EnumChildWindows | Create a list of child windows of a parent window. |
| SetWindow | Lets you maximize, minimize, or restore the PIXCL window. |
| UseCaption | Lets you set the text that appears in the title bar of the PIXCL window. |
| WinClose | Closes any specified window by title string. |
| WinExist | Determines whether a parent window is running. |
| WinGetActive | Returns the name of the foreground window. |
| WinGetClientRect | Returns the coordinates of the specified window client area. |
| WinGetLocation | Gets the location a parent window and returns the specified coordinates. |
| WinHelp | Starts the Windows Help utility and displays the requested Help file and topic. |
| WinAdjustRect: | Returns the window coords for a specified client area. |
| WinLocate | Locates a parent window at the specified coordinates. |
| WinSetActive | Activates a running application. |
| WinShow | Hides, unhides, minimizes, maximizes, or restores a specified window. Can also set any window to topmost or not topmost. |
| WinTitle | Sets the text that appears in the title bar of a window. |

Pausing a Program

In PiXCL, you can pause a program a specified number of seconds, or you can pause it indefinitely. Both require the [WaitInput](#) command.

Pausing a Specified Number of Seconds

By using the [WaitInput](#) command with an argument, you can pause a PiXCL program a specified number of seconds. For example, the following program displays the message "Waiting...". Next, it pauses the program for three seconds and then erases the window's contents.

```
DrawText(10,10,"Waiting...")
WaitInput(3000) {Pause the program for 3 seconds}
DrawBackground {Erase the window's contents}
WaitInput()
```

While the parameter for the [WaitInput](#) command is in milliseconds, PCs have a timer granularity effect, that is, the minimum tick of the system clock is actually 64 milliseconds, so [WaitInput\(1\)](#) is in effect the same as [WaitInput\(64\)](#).

Pausing Indefinitely

By using the [WaitInput](#) command without an argument, you can pause a PiXCL program indefinitely. In general, you should use [WaitInput\(\)](#) whenever you are not performing any work in your PiXCL window, because it makes more system resources available to other programs.

If you fail to use [WaitInput\(\)](#) at some point in your program, one of two things will happen: either the window will disappear immediately after executing your script or, if you are stuck in a continuous loop, the hour-glass icon will always be present.

The [WaitInput\(\)](#) command is also important for building your own custom menus and 3-D command buttons, and for getting keyboard or mouse input. The examples in the next few sections will give you more of a feel for how [WaitInput\(\)](#) works. It is often necessary to pause the program so that Windows can catch up and update other windows and background, by using the [WaitInput\(100\)](#) command.

Interprocess Communications

One PiXCL application can also contain a [PXLResume](#) command that sends a message to another PiXCL application that is paused indefinitely with a [WaitInput\(\)](#) command, and it will resume operation. The [PXLResumeAt](#) command extends this capability and lets one PiXCL application (or any other application that has been programmed appropriately) tell another PiXCL application to start execution at a specific label. This can be very useful if you want several PiXCL applications to run at the same time and respond to input received from each other.

For example, you can have several applications running that are processing different aspects of an image data set. An additional PiXCL application could be the controller that selects processing or pen and brush functions. Once a new function has been selected, an INI file or registry setting could be made, and the secondary applications all instructed to read the updated control parameters, and commence processing. The controller application would then wait until each secondary application had sent a message (via a [PXLResumeAt](#) message) that processing was complete.

A useful additional interprocess command is [GetCopyDataMsg](#) that is used when another application (PiXCL runtime or user program in C or C++) needs to send a data string such as a filename to be loaded.

PiXCL also acts as a dropfile server and drop file client, such that you can drag and drop text and image files into a PiXCL window (client mode) or drag from a PiXCL window to another application client window (server mode). See the [DropFileServer](#), [DragAcceptFile](#) and [GetDragList](#) commands for more information.

Building Menus

To build your own custom menus in PiXCL, you use the [SetMenu](#) command to define a menu template. Then, when the program is pausing for input (a [WaitInput\(\)](#) command is in effect) and the user selects a menu item, the program branches to the label associated with that menu item, as defined in the template.

PiXCL also supports floating popup menus invoked with a right mouse click, via the [SetPopupMenu](#) command.

For example the following program creates a simple menu with two menu items: Write and Exit!. If you select the Write option, the program branches to the Run_write label where Windows write is launched. If you select the Exit! option, the program ends.

```
{Define the menu template}
    SetMenu("Write",Run_write,
    ENDPOPUP,
    "Exit!",Leave,
    ENDPOPUP)
Wait_for_input:
    WaitInput()
Run_write:
    Run("WRITE.EXE")
    Goto Wait_for_input
Leave:
    End
```

Because the [SetMenu](#) command uses strings enclosed in quotes, you can also use string variables (e.g. [Menu_Item\\$](#)) in the command. This can be very useful when you need to dynamically change the text in a menu, without having to write a new [SetMenu\(...\)](#) command in your script.

If you use the [SetMenu\(\)](#) command without any arguments, it removes the menu bar from the Window. The first time that you use the [SetMenu\(\)](#) command, it must be immediately preceded with a [WaitInput\(100\)](#) command. This is to allow Windows to catch up and complete the redrawing of the new PiXCL application window. i.e.

```
WaitInput(100) {let NT and 95 catch up}
SetMenu(...)
```

If you don't do this, the screen will often flash when Windows attempts to redraw the client area.

Changing Menu Characteristics

PiXCL has two commands for changing the appearance of menu bar pop-up items: [ChangeMenuItem](#) and [GetMenuStatus](#). The [ChangeMenuItem](#) command checks, unchecks, grays, or enables a main menu bar item: they have no effect on popup menu items created with the [SetPopupMenu](#) command. Building on the previous example, suppose you want to place a checkmark next to the Write option immediately after you launch Windows Write. Here's the command you would use immediately after

```
Run("WRITE.EXE")
ChangeMenuItem("Write",CHECK,Result)
```

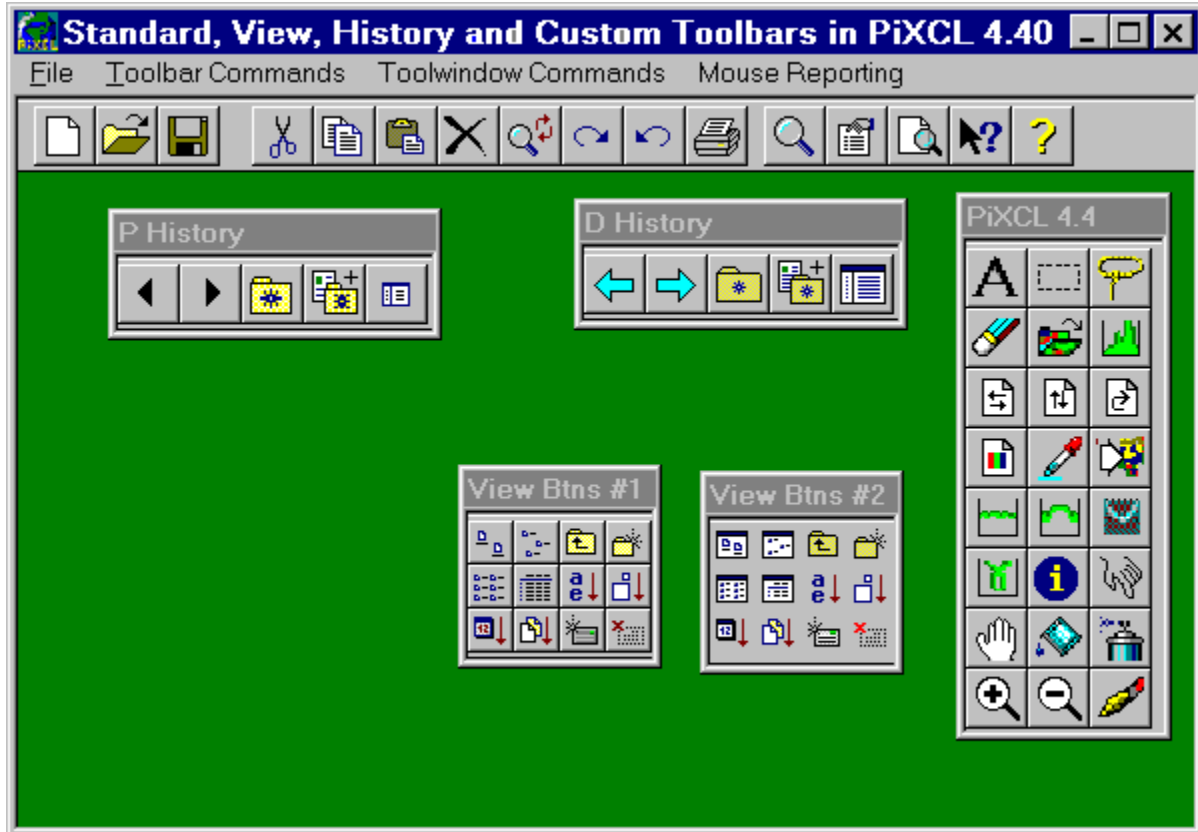
Because you may want to examine a menu item's appearance before changing it, PiXCL offers the [GetMenuStatus](#) command. For example, this sequence checks to see whether the Write option is grayed. If so, the program enables the menu item (removes the gray) before placing a checkmark next to it.

```
GetMenuStatus("Write",GRAYED,Result)
```

```
If Result = 1 Then ChangeMenuItem("Write",ENABLE,Result)
ChangeMenuItem("Write",CHECK,Result)
```

Building ToolBars and ToolWindows

PIXCL supports a user definable and customizable toolbar with automatic tooltips that provides fast equivalents to menu item selections, or functions that you do not want to define as a menu item. There is a set of standard toolbar buttons that are built into the Windows Common controls. Additional button bitmaps are built into PIXCL itself, in the same way that there is a set of icon images.



The [ToolWindow](#) command has a similar syntax and function to the [ToolBar](#) command, but creates an arbitrary number of floating button windows, in POPUP mode that creates a toolwindow that floats anywhere in the screen.

A toolbar or toolwindow can be raised or flat style, and can each have up to 76 buttons, in a variety of initial states and styles. Once a toolbar is displayed, buttons can be moved around and deleted without recreating the toolbar by double clicking on the toolbar background. The image above shows the set of large size Standard toolbar with groups of buttons, plus floating toolwindows that show the large size View and PIXCL built-in buttons, plus the small size History buttons.

Toolbars and toolwindows are used to provide buttons that

- equate to menu shortcuts for commonly used selections
- provide a set of paint and draw selections for which a menu item may not be appropriate.

Changing Toolbar and ToolWindow Button Characteristics

You can query and update the state of any button in a toolbar or toolwindow with the [GetToolBarBtnStatus](#) and [ChangeToolBarBtn](#) commands. Toolbars and toolwindows can be customized with the [CustomizeToolBtn](#) command, or by double clicking on a toolbar or toolwindow background. If you need to change button bitmaps or tooltips, you must re-issue the [ToolBar](#) or [ToolWindow](#) command.

3-D Command Buttons

PiXCL's [Button](#) command lets you create standard Windows 3-D pushbuttons, radio buttons, check boxes and group boxes, and place them anywhere within the PiXCL window. When the program is pausing for input and the user clicks on a button, the program branches to the label associated with that button. The figure below shows a variation of the previous menu example with some buttons added to mimic the actions of the menu items.

Buttons are active for the left mouse only.

Here's the code used to produce the example:

```
{Define the menu template}
    WaitInput(100)
    SetMenu("Write",Run_write,
           ENDDPOPUP,"Exit!",Leave,
           ENDDPOPUP)

{Draw 3-D buttons}
    Button(20,20,60,35,"Write",Run_write,
          20,45,60,60,"Exit",Leave)

Wait_for_input:
    WaitInput()

Run_write:
    Run("WRITE.EXE")
    Goto Wait_for_input

Leave:
    End
```

As the example shows, PiXCL automatically provides mouse support for 3-D command buttons. If you want to test for mouse clicks in other areas of the PiXCL window, you'll need to use the [SetMouse](#) command, described next. If you want the user to be able to select a 3-D button using the keyboard, you'll need to use the [SetKeyboard](#) command (see "[Getting Keyboard Input](#)" later).

During development testing on some early Windows 95 machines, we often had problems with programs hanging when a [Button](#) command was reached. This was usually fixed by loading the latest video device driver. If you are having problems on a Windows 95/98 or NT4 PC, please check with your PC supplier or video card vendor for the latest driver. If you still have problems, please contact VYSOR Technical Support at <http://www.vysor.com>.

3D Command Buttons with BitMaps

In PiXCL 5.0 and later, the [Button](#) command supports the specification of BMP or RLE images and ICO icon files on the button, instead of a text string. In addition, you can specify any of the PiXCL 20 or so built-in icons plus around another 70 icons from the Windows SHELL32.DLL. Buttons are active for the left mouse only.

Alternatively, if you want buttons that support middle and right mouse clicks, with a bit of ingenuity with the [DrawBitmap](#) command and one or more of the [SetMouse](#) commands, you can create 3-D command buttons with bitmaps and place them anywhere within the PiXCL window. When the program is pausing for input and the user clicks on a BitMap button (actually a mouse active region), the program branches to the label associated with that mouse hit.

The bitmaps you use should ideally be 16 color, as this minimizes the amount of data to read off the disk (or from the system disk cache if it exists). When making "pushed" bitmaps, the traditional style is to move the bitmap design right and down one pixel. You can also use the [InvertRectangle](#), that negates the bitmap colors to indicate that a bitmap region had been selected with the mouse.

```
{Define the button Bitmaps}
    ButtonPos$ = "btn_pos.bmp" {32x32 16-color}
    ButtonNeg$ = "btn_neg.bmp" {32x32 16-color}
    DrawBitMap(20,20,ButtonPos$)
{set up the mouse hit regions}
    SetMouse(20,20,51,51,Run_write,X,Y)
    SetMenu("Exit",Leave,ENDPOPOP)
Wait_for_input:
    WaitInput()

Run_write:
    DrawBitMap(20,50,ButtonNeg$)
    WaitInput(250) {click in the button and wait}
    DrawBitMap(20,50,ButtonPos$)
    Run("WRITE.EXE")
    Goto Wait_for_input
Leave:
    End
```

You can also use any of the icons built into PiXCL, as well as the system defined [MessageBox](#) icons as images in buttons. The method is essentially the same as shown above, except the [DrawBitmap](#) command is replaced by a [DrawIcon](#) command. Other related commands that are useful to create pushbuttons with images are [DrawEdgeRectangle](#) and [InvertRectangle](#).

The icons built into PiXCL and hence a PiXCL runtime can also be changed to whatever you require, using an icon management tool.

Getting Mouse Input

To get mouse input in a PiXCL program outside of 3-D command buttons, you use one or more of the [SetMouse](#), [SetShftMouse](#), [SetCtrlMouse](#), [SetDbI Mouse](#), [SetRightMouse](#), [SetCtrlRightMouse](#) [SetShftRightMouse](#) or [SetDbIRightMouse](#) commands and define rectangular regions on the screen as mouse hit-testing regions. If you have a Microsoft Intellimouse™ or similar, [SetMidMouse](#), [SetShftMidMouse](#), [SetCtrlMidMouse](#) or [SetDbIMidMouse](#) commands are also available. Then, when the program is pausing for input and you click the mouse within a mouse hit-testing region, the program branches to the label associated with that region, as defined by the [SetMouse](#) command. The argument syntax is the same for all twelve mouse commands. Mouse active areas can be overlapped. [Set\[*\]Mouse](#) and [SetDbI\[*\]Mouse](#) should generally NOT be overlapped, as the [Set\[*\]Mouse](#) command will catch the area first. All the double mouse commands can be overlapped.

You can also define polygon shaped mouse active regions by creating a set of overlapped or adjacent rectangles that cover the desired polygonal region.

For example, suppose you want to modify the previous program to display a warning message when the user clicks within the window, but misses a 3-D button. Here's the code to accomplish this:

```
{Define the menu template}
    SetMenu("Write",Run_write,
    ENDPOPOP,
    "Exit!",Leave,
    ENDPOPOP)

{Draw 3-D buttons}
    Button(20,20,60,35,"Write",Run_write,
    20,45,60,60,"Exit",Leave)

{Use pixel coordinates for mouse hit testing; more accurate}
    UseCoordinates (PIXEL)

{Set mouse hit-testing region to entire screen}
    GetScreenCaps(HORZRES,x)    {Get screen's vert. resolution}
    GetScreenCaps(VERTRES,y)    {Get screen's horiz. resolution}
    SetMouse(0,0,x,y,Missed,Temp,Temp) {Branch to Missed on miss-hit}

Wait_for_input:
    WaitInput()

Run_write:
    Run("WRITE.EXE")
    Goto Wait_for_input

Leave:
    End

{Put up message box when user misses 3-D button with mouse}
Missed:
    MessageBox(OK,1,EXCLAMATION,
    "Click on a button or select a menu item!",
    "You missed...",Temp)
    Goto Wait_for_input
```

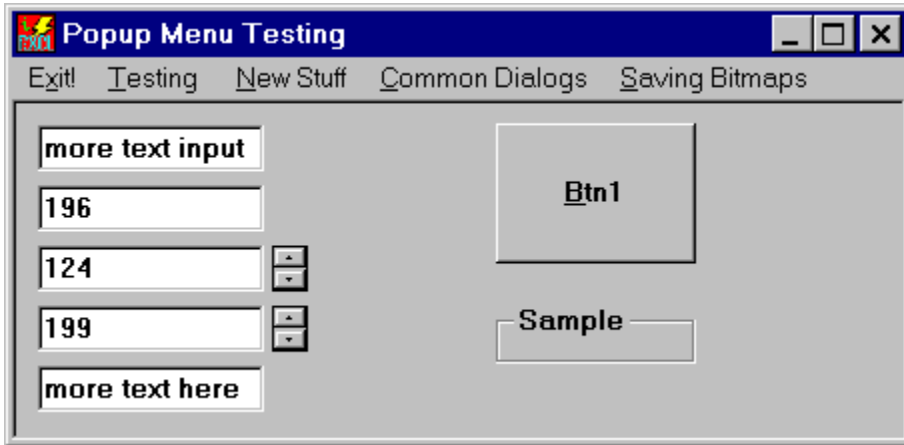
In this example, the entire screen is set up as a hit-testing region by the [SetMouse](#) command. By using the [GetDeviceCaps](#) command, the program determines the pixel resolution of the screen driver. For example, if you are using a SuperVGA driver, the hit-testing region is defined as (0,0) to (1023,767). If you click the mouse within this region but miss a 3-D button, the program

branches to the [Missed](#) label where a message box is displayed telling you to click on a button.

Getting Keyboard Input

PIXCL provides you with three ways to input data from the keyboard. Firstly, if there are multiple string or numeric data that potentially needs editing on entry, via the [SetEditControl](#) command, secondly, using a [TextBox](#) dialog for single lines of text or numeric data, and thirdly when there are a set of keystrokes (e.g. accelerator keys for user defined functions and buttons) that are required via the [SetKeyboard](#) command.

The [SetEditControl](#) command is similar to the [Button](#) command, and is generally used in conjunction with it to create client area dialogs. [SetEditControl](#) creates an arbitrary number of single line edit windows of arbitrary size, with text displayed in the currently selected font. Edit controls can be used to enter string variable characters, without having to resort to the [SetKeyboard](#) method described below.



Example of multiple edit controls using the [SetEditControl](#) and [Button](#) commands.

To read single keystrokes in a PIXCL program, you use the [SetKeyboard](#) command to define the keys that you'll accept. Then, when the program is pausing for input and you press a specified key, the program branches to the label associated with that key.

Syntax:

[SetKeyboard\(\)](#) This clears all previous [SetKeyboard](#) commands.

or

[SetKeyboard\("a",Label,"^a",Label,vkey,Label\)](#)

Parameters:

- ["a"](#) Any white key on the keyboard (except function keys and certain keys on the numeric keypad). For example, "a" represents lowercase **a** and "R" represents uppercase **R**.
- ["^a"](#) Any white key on the keyboard (except function keys and certain keys on the numeric keypad) in combination with CTRL. For example, "^b" represents CTRL+b and "^V" represents CTRL+V.
- [vkey](#) A virtual key number taken from the Table below. For example, the virtual key number for the F1 function key is 112. Using a virtual key number is the only way to test for certain keys, including function keys and several keys on the numeric keypad.
- [Label](#) A label you want PIXCL to branch to when the user presses the preceding key. For example, the command [SetKeyboard\("C",Run_calc\)](#) causes the program to branch to the label Run_calc when the user presses C.

| Value | Description | Value | Description | Value | Description |
|-------|--|-------|--|-------|--|
| 8 | BACKSPACE | 9 | TAB | 12 | 5 on numeric keypad with NUMLOCK off |
| 13 | ENTER | 16 | SHIFT | 17 | CTRL |
| 18 | ALT | 19 | PAUSE (or CTRL + NUMLOCK) | 20 | CAPS LOCK |
| 27 | ESCAPE | 32 | SPACEBAR | 33 | PGUP |
| 34 | PGDN | 35 | END | 36 | HOME |
| 37 | LEFTARROW | 38 | UPARROW | 39 | RIGHTARROW |
| 45 | INSERT | 40 | DOWNARROW | 44 | PRINTSCREEN |
| 49 | 1 | 46 | DELETE | 48 | 0 |
| 52 | 4 | 50 | 2 | 51 | 3 |
| 55 | 7 | 53 | 5 | 54 | 6 |
| 65 | A | 56 | 8 | 57 | 9 |
| 68 | D | 66 | B | 67 | C |
| 70 | F | 69 | E | 72 | H |
| 73 | I | 71 | G | 75 | K |
| 76 | L | 74 | J | 78 | N |
| 79 | O | 77 | M | 81 | Q |
| 82 | R | 80 | P | 84 | T |
| 85 | U | 83 | S | 87 | W |
| 90 | Z | 86 | V | 89 | Y |
| | | 88 | X | 97 | Numeric key pad 1 (NUMLOCK must be on) |
| 98 | Numeric key pad 2 (NUMLOCK must be on) | 96 | Numeric key pad 0 (NUMLOCK must be on) | 100 | Numeric key pad 4 (NUMLOCK must be on) |
| 101 | Numeric key pad 5 (NUMLOCK must be on) | 99 | Numeric key pad 3 (NUMLOCK must be on) | 103 | Numeric key pad 7 (NUMLOCK must be on) |
| 104 | Numeric key pad 8 (NUMLOCK must be on) | 102 | Numeric key pad 6 (NUMLOCK must be on) | 106 | Numeric key pad * |
| 107 | Numeric key pad + | 105 | Numeric key pad 9 (NUMLOCK must be on) | 110 | Numeric key pad . (NUMLOCK must be on) |
| 111 | Numeric key pad / | 109 | Numeric key pad - | 113 | Function Key F2 |
| 116 | Function Key F5 | 112 | Function Key F1 | 115 | Function Key F4 |
| 119 | Function Key F8 | 114 | Function Key F3 | 118 | Function Key F7 |
| 122 | Function Key F11 | 117 | Function Key F6 | 121 | Function Key F10 |
| 125 | Function Key F14 | 120 | Function Key F9 | 124 | Function Key F13 |
| 144 | NUM LOCK | 123 | Function Key F12 | 127 | Function Key F16 |
| | | 126 | Function Key F15 | | |
| | | 145 | SCROLL LOCK | | |

The following key codes apply to US keyboards only:

- 186 Colon/semi-colon
- 187 Plus/equal

| | |
|-----|--|
| 188 | Less than/comma |
| 189 | Underscore/hyphen |
| 190 | Greater than/period |
| 191 | Question/slash |
| 192 | Tilde/backwards single quote |
| 219 | Left curly brace/left square brace |
| 220 | Pipe symbol/backslash |
| 221 | Right curly brace/right square bracket |
| 222 | Double quote/single quote |

Virtual Key Numbers Table

Suppose you want to add keyboard support to the previous example so that when you press **w** or **W** the program runs Write and when you press **e** or **E**, the program exits. Here's how you would modify the program:

```
{Define the menu template}
    SetMenu("Write",Run_write,ENDPOPUP,
           "Exit!",Leave,
           ENDPOPUP)

{Draw 3-D buttons}
    Button(20,20,60,35,"&Write",Run_write,
          20,45,60,60,"&Exit",Leave)

{Set up keyboard support for buttons}
    SetKeyboard("W",Run_write,
              "w",Run_write,
              "E",Leave,
              "e",Leave)

{Use pixel coordinates for mouse hit testing; more accurate}
UseCoordinates(PIXEL)

{Set mouse hit-testing region to entire screen}
    GetScreenCaps(HORZRES,x)    {Get screen's vert. resolution}
    GetScreenCaps(VERTRES,y)   {Get screen's horiz. resolution}
    SetMouse(0,0,x,y,Missed,Temp,Temp)
        {Branch to Missed on miss-hit}

Wait_for_input:
    WaitInput()

Run_write:
    Run("WRITE.EXE")
    Goto Wait_for_input

Leave:
    End

{Put up message box when user misses 3-D button with mouse}
Missed:
    MessageBox(OK,1,EXCLAMATION,
              "Click on a button or select a menu item!",
```

```
"You missed...",Temp)  
Goto Wait_for_input
```

Note#1: It is also possible to use a [SetKeyBoard\(\)](#) command to catch a large number of key codes, and write directly into an edit dialog region in the client area. See the sample program [keyboard.pxI](#) for one way of doing this.

Sound Card Support

PiXCL provides a set of commands to play WAV files on standard SoundBlaster™ compatible sound cards. WAV files can be played synchronously or asynchronously, and volume, pitch and playrate are adjustable.

Using WAV files can add useful help or other audible prompting to your applications, and are commonly used in multimedia presentations.

One of the ways that sound can be used in to create a WAV file of say, introductory music and speech that runs for a set length of time, and play it asynchronously. The PiXCL application continues while the WAV is playing, and at set times, you can update the screen with new text and images. Careful use of the WaitInput command ensures that your application plays more or less the same on systems with a variety of cpus and cpu clock speeds.

PiXCL 5.0 and later also support the set of Multimedia Control Interface (MCI) commands that many devices support. MCI commands can be used to play overlapping audio files at the same time.

Serial Communications with PiXCL

PiXCL provides programmed read/write access to your PC serial ports, COM1, COM2, COM3 and COM4, for purposes such as reading data streams from input devices like digitising tables, and commanding or reading devices that are controlled from an RS-232 port. These can be as diverse as smart home appliances to model trains.

Most PCs have two serial ports only, COM1 and COM2, with COM2 often assigned to a modem for Internet access. Some PCs use COM1 for the mouse input, while most newer PCs have a PS2 style mouse port, which leaves COM1 available for other devices.

The PC design supports four standard serial ports that share only two interrupt lines. Hence, COM1 and COM3 share one, and COM2 and COM4 share the other. This means that if you want install two serial cards and use two serial devices on COM1 and COM3, your software will have to be able to support shared interrupts.

COM port support in PiXCL is not a device driver, and does not support shared interrupts.

Communications Commands:

[ClearCommPort](#) [GetCommPort](#) [ReadCommPort](#) [SetCommPort](#) [WaitCommEvent](#) [WriteCommPort](#)

Importing images from TWAIN-compliant devices

PIXCL provides command support for TWAIN compliant data source devices. These are typically scanners, digital still and video cameras, and some photocopiers with a scanner output function.

Both 16 and 32 bit data sources are supported in TWAIN v1.6. Please note that in TWAIN v1.7 (released September '97), support for 16 bit devices has been dropped. See the TWAIN Working Group site at <http://www.twain.org> for details.

There are hundreds of TWAIN devices on the market, some with only 16 bit drivers, some with both 16 and 32 bit drivers. We have found that some of the older 16 bit devices do not have very well behaved drivers, and can cause a 32 bit program trying to acquire an image to crash. We very strongly suggest that

- a) you locate the latest TWAIN driver from the device supplier, if 16 bit;
- b) you locate a TWAIN 32 bit driver for the device in preference to a 16 bit driver; and
- c) if the device is more than five years old, or out of production, that purchase of a new device is justifiable.

Support for TWAIN devices requires the following four files to be present in your Windows directory: TWAIN.DLL and TWAIN_32.DLL, and TWUNK_16.EXE and TWUNK_32.EXE. The last two provide the necessary conversion between 16 bit TWAIN data sources and 32 bit applications such as PIXCL. These four DLLs are provided with PIXCL, and are installed if necessary. In addition, PIXCL 4.14 and later also provide PXLtwain.DLL, which is the command interface between PIXCL and TWAIN_32.DLL. PXLtwain.DLL is also stored in the [windows](#) directory.

If you have a file TWAIN32.DLL (note no “_”) in your [windows](#) or [windows\system](#) directory, it should be deleted, as it is a beta release library, and is no longer valid for Windows 95/98 or NT 4.0.

In addition, your TWAIN device will come with one or more .DS files. These are the binary drivers (actually DLLs with a DS file extension) that TWAIN_32.DLL needs to work. 16 bit .DS files are stored in your [windows\twain](#) directory, and 32 bit .DS files in the [windows\twain_32](#) directory.

PIXCL provides twenty-five commands to control the TWAIN device. You can acquire images and write them to the PIXCL image list, copy the image to the Clipboard directly, or write the image to a file.

See the sample program **twaindev.pxl** for examples of using the TWAIN commands.

Running Other Programs

By using the `Run (...)` and `RunExt (...)` commands you can execute other applications from within a PiXCL program. For example, the following program starts Notepad and then launches a copy of a command interpreter, CMD.EXE:

```
Run ("NOTEPAD.EXE")
Run ("C:\APPS\CMD.EXE")
WaitInput ()
```

Each program you start takes on a life of its own independently of the PiXCL script that invoked it. This means that PiXCL does not pause after executing a `Run` command, but continues with the next command in the script. For example, in the previous program, PiXCL starts Notepad then immediately starts CMD.EXE.

If PiXCL could not run the specified program, four error conditions are trapped, as follows:

- the PATH specified is invalid or does not exist.
- the EXE file cannot be found.
- the EXE file is corrupted or unreadable.
- Windows does not enough memory or resources available.

The last error is unlikely to occur with commercial release version programs. If the specified file cannot be run, a `MessageBox` appears informing you of the name of the EXE file, and the apparent reason that the file could not be run. The script will continue processing in all cases, so it is up to you to decide what to do if the program does not run. The most useful check is to verify that the new window created by the EXE actually exists, using the `WinExist` command.

Using the Windows Shell functions with PiXCL

PiXCL has two commands, [ShellAbout](#) and [FindExecutable](#), that use Windows shell functions directly, but there are more functions available with a bit of simple programming.

The files `c:\windows\Rundll32.exe` (Windows 95/98) and `c:\winnt\system32\Rundll32.exe` (Windows NT) are used extensively by Windows 95 and NT4 to launch a wide range of actions defined in DLLs. For example, many of the Control Panel applets are defined in a series of `.cpl` files, and can be run as part of a PiXCL program, using a command line passed to the [Run](#) or [RunExt](#) command. These applets all produce a dialog that can be used as necessary in your PiXCL applications.

If you do a search in `c:\windows\system` you will find 16 or so `.cpl` files. These are binary files (actually DLLs with specific entry functions), but there are ways to investigate the contents. An easy way is to use the Start:Run window on the Taskbar, with the command line syntax shown below.

The general PiXCL Run command syntax of the shell command line is

```
Run("path\rundll32.exe shell32.dll,Control_RunDLL filename.cpl,@number[,page_number]")
```

or alternatively

```
Run("path\control.exe filename.cpl,@number[,page_number]")
```

where

`@number` the 0-based ordinal for a multi-function applet.
`page_number` the optional sometimes 1 and sometimes 0-based property sheet number on a multi-page applet dialog. Its SUPPOSED to be 1-based according to Microsoft. You'll need to experiment.

For example, if you right click the Windows 98 screen background and select Properties, you get the **Display Properties** dialog, typically with tabs named **Background**, **ScreenSaver**, **Appearance**, **Effects**, **Web** and **Settings**, which are numbered left to right 0,1,2,unknown, unknown,3. Hence, to display the Settings tab, the command would be

```
Run("path\control.exe desk.cpl,@0,3")
```

Note that there are no spaces in `shell32.dll,Control_RunDLL` and `filename.cpl,@number[,page_number]`. Note also that `Control_RunDLL` is case sensitive, as it's the entrance point into the shell dll.

Here is a list of some of the Control Panel Applets you will find on your system. The list is not exhaustive, and your system may well be different.

AppWiz.cpl

0 Remove/Install applications

Desk.cpl

0 Display Properties (multi-page)

FindFast.cpl

0 FindFast dialog

Inetcpl.cpl

0 Internet Properties (multi-page)

Intl.cpl

0 Regional Settings Properties (multi-page)

Joy.cpl

0 Joystick Properties

Main.cpl

0 Mouse Properties (multi-page, but sheet indices don't work)

1 Keyboard Properties (multi-page)

2 Printer Icons window

3 Fonts Icons window

ML32cfg.cpl

0 MS Exchange Settings Properties (multi-page)

Mmsys.cpl

0 MultiMedia Properties (multi-page)

1 Sounds Properties (multi-page)

Modem.cpl

0 Modems Properties (multi-page)

Sysdm.cpl

0 System Properties (device manager) (multi-page)

1 Add New Hardware Wizard

Timedate.cpl

0 Date / Time Properties

Another shell command allows you to bring up the format floppy disk dialog. The PiXCL command is [Run\("path\rundll32.exe shell32.dll,SHFormatDrive"\)](#)

You can display the "Open With" dialog with the PiXCL command. The dialog displayed is slightly different if the [<filename.ext>](#) is included in the command line.

[Run\("path\rundll32.exe shell32.dll,OpenAs_RunDLL <filename.ext>"\)](#)

Invoking on-line Help Files

You can invoke any Windows Help file (*.HLP) with either the PiXCL [Run \(...\)](#) command or [WinHelp \(...\)](#) command. The standard Help file viewer engine is [c:\windows\winhlp32.exe](#), in Windows 95/98 and [c:\windows\system32\winhlp32.exe](#) in Windows NT i.e. an standard executable file. PiXCL 4.22 and later also support the new compiled HTML (.CHM) format that comes with Windows 98 and later, with the [WinHTMLHelp](#) command.

For example, the following code fragments start WINHLP32 with the argument to display a Help file.

```
Run ("WINHLP32 newapp.hlp")  
WaitInput ()
```

and

```
WinHelp ("newapp.hlp", "Contents", "")  
WaitInput ()
```

are functionally equivalent

The [.EXE](#) extension is not necessary as Windows assumes this is correct. It is also not necessary to include a PATH with WINHLP32, as Windows already knows where it is.

See the complete details in the sections on the [Run](#) command and the [WinHelp](#) and [WinHTMLHelp](#) commands.

Printing documents and images with PiXCL

There are several ways to print a document in Windows. The way you are likely most familiar with is starting a document editor like NotePad, Write or WordPad, or an image editor like Paintbrush, opening the desired file then selecting the Print menu option. Most Windows utilities and many third party applications also provide a command line print argument.

For example, if you invoke NotePad with a `/p` argument, it will open the selected document, start the Print Manager, print the file, then close NotePad.

In a PiXCL program, you can use the `Run` or `RunExt` commands to print a document or image. The command syntax could be

```
Run("NotePad /p textfile.txt")
```

or

```
PrintJob$ = "NotePad /p textfile.txt"  
RunExt(PrintJob$,NORMAL,"",WinDir$, 0,0,0,0, NOWAIT,0)
```

You can also locate the executable file associated with a particular file type with the `FindExecutable` command. For example, if your PiXCL application works with a TIF file, on one PC the associated EXE file may be Paint Shop Pro™, and on another ThumbsPlus™. The `FindExecutable` returns the correct application to print the file, and this is plugged into the `Run` or `RunExt` command as the first argument.

If you want to print a large file, you could set the NotePad windows to run minimized, using the `EnumWindows` and `WinShow` commands. You can also set the printing application window (defaults of 0,0,0,0 above) to negative numbers i.e. outside of the viewable area.

The following Windows utilities support command line printing:

[NotePad](#), [Write](#), [Paintbrush](#), [CardFile](#)

For printing documents with other applications, check in the application documentation or the manufacturer. Most new 32 bit applications also write information into the Windows Registry that lists the arguments needed for printing documents. You can access the registry with the `RegEdit` program which is purposely hidden by Microsoft in the Windows directory. Search in the **CLASSES** tree for the relevant program name, then look in the **shell** branch for the references to print commands.

PiXCL also provides full read and write access into the Windows Registry. See [Access to the Windows Registry](#) for more information.

The second method is to use the `PrintFile` command that is built into PiXCL. This is similar to the above method, but does it all in one command. Windows supports a so called shell processor that provides document printing support. The shell looks at the document type (e.g. INI, TXT, DOC, BMP and so forth), looks up the Registry for the necessary application that is indicated as being able to print the document, starts the application in the background and prints it. Finally, the printing application is closed automatically. `PrintFile` is a very handy command, as it enabled you to print any document if the application is installed on your PC.

Please note that `PrintFile` is not meant to provide comprehensive document and imaging print services. If you need to print, for example, a Microsoft Word™ document, or a large CORELdraw™ vector file, then the `Run(...)` method or `PrintFile(...)` method described above is the appropriate choice. `PrintFile` is designed for text and other document type files.

You can also print any of the supported bitmap formats with the `PrintBitmap` command. With this command, you have the option of displaying the PageSetup and Print common dialog boxes.

Managing Files and Directories

PiXCL has a wide range of commands that let you manage files and directories, as shown in the table below. For example, here is how you use the [FileCopy](#) command to copy all the files in the C:\LETTERS directory with a .TXT extension to the root directory of drive A:

```
FileCopy("C:\LETTERS\*.TXT", "A:\", Copied_Files)
```

Here the [Copied_Files](#) variable reports the number of files successfully copied.

| <u>Command</u> | <u>Purpose</u> |
|-------------------------------|---|
| DirChange | Makes a specific directory the current directory. |
| DirExplore | Created as Explorer window that lists the current directory contents. |
| DirGet | Gets the current directory. |
| DirGetSystem | Gets the Windows system directory. |
| DirGetWindows | Gets the Windows directory. |
| DirMake | Creates a new directory. |
| DirRemove | Deletes an existing directory. |
| DiskChange | Makes another disk drive current. |
| FileCopy | Copies one or more files from one directory to another. |
| FileDelete | Deletes one or more files from disk. |
| FileExist | Tests whether a file exists. |
| FileExtension | Extracts the file extension from a filename string. |
| FileGetDate | Reads the date stamp in a file's directory entry. |
| FileGetSize | Reads the size in bytes of the specified file. |
| FileGetTime | Reads the time stamp in a file's directory entry. |
| FileMove | Moves one or more files from one directory to another. |
| FileName | Extracts the rootname from a filename string. |
| FilePath | Extracts the path from a filename string. |
| FileRead_INI | Read section and key strings from any INI file |
| FileRename | Changes the name of one or more files. |
| FileWrite_INI | Write section and key strings to any INI file. |
| GetDiskSpace | Returns the disk type, total space and free space. |

As another example, here's how you can use the [DirGet](#) commands to place the source and current directory in the [SourceDir\\$](#) and [CurrentDir\\$](#) string variables:

```
DirGet(SourceDir$)
... some FileGet commands or similar ...
DirGet(CurrentDir$)
```

This command is useful to locate the starting directory of a program and store it in a string variable, then change to another directory. This is analogous to the **Program Manager: File Properties** "Command Line" and "Working Directory" entries.

PiXCL 4.0 and later also support the use of initialization or INI files to store application parameters. INI files should always be created and edited with NotePad. PiXCL commands [FileRead_INI](#) and [FileWrite_INI](#) provide access to any INI file in your computer, including SYSTEM.INI and WIN.INI.

An INI file for a PiXCL application could include such information as

- program startup display coordinates
- default background color
- last time a certain file was accessed
- a bitmap to be displayed

More than one INI file can be accessed by your PiXCL application. For example, if you have a PiXCL program that works in conjunction with another application called BMPDSPLA.EXE that, say, keeps track of a working directory, and both applications need to work from that same directory, using the [FileWrite_INI](#) command to change BMPDSPLA.INI allows either application to set the working directory.

Any information that can be stored in an INI file can be written into the Windows Registry, also known as the Registration Database. Microsoft suggests that all new 32-bit applications make use of the Registry rather than using INI files. PiXCL provides a set of commands starting with RDB (i.e. Registration DataBase) to read and write the Registry.

See also [Access to the Windows Registry](#) for more information.

Clipboard Operations

PiXCL has three commands for reading text from and writing ASCII text to the Windows Clipboard, plus a command to empty the Clipboard.

| | | |
|------------------------------|---|---|
| <code>ClipboardAppend</code> | — | Adds text to the end of the Clipboard's existing text. |
| <code>ClipboardGet</code> | — | Reads text from the Clipboard. |
| <code>ClipboardPut</code> | — | Copies text to the Clipboard, replacing Clipboard's current contents. |
| <code>ClipboardEmpty</code> | — | Clears the Clipboard of text and images and other binaries. |

The Clipboard functions are extremely useful for passing text information between programs. For example, a control program can pass a set of directories or filenames to subsidiary programs.

PiXCL supports some binary image data access to the Clipboard too. For example, the [TWAIN_AcquireToClipboard](#) command will copy a bitmap from a scanner type device, to the clipboard.

Images can be copied to and from the clipboard with the

| | | |
|------------------------------------|----|--|
| <code>ClipboardGetBitmap</code> | — | Gets a bitmap into the PiXCL image list in memory. |
| <code>ClipboardPutBitmap</code> | -- | Copies a bitmap from the image list. |
| <code>CopyWindowToClipboard</code> | -- | Copies a selected window to the clipboard i.e. window capture. |

Creating and using INI files

PiXCL provides two commands, [FileRead_INI](#) and [FileWrite_INI](#) to access any system or application initialization file. For example, you may need to invoke a paint and draw program with a standard set of startup parameters like working image and saving image directories, default position or file type. A PiXCL program with perhaps 20 to 50 lines of code can be created to access these desired parameters in your own INI file, and update the application INI file before running the application. If the [DropFileServer](#) command is used, you can even select a group of files of various formats and have the application load them sequentially.

PiXCL also provides access to the Windows Registration Database that is used to store information in the same way as INI files.

See also [Access to the Windows Registry](#) for more information.

Access to the Windows Registry or Registration Database

You are probably aware of and used to using Windows application initialization files, or INI files as they are more commonly known. These are described in moderate detail in the section of this manual that references the [FileRead_INI](#) and [FileWrite_INI](#) commands.

INI files are very useful for storing a wide variety of application and system parameters, but with the advent of 32 bit Windows and more particularly Microsoft Object Linking and Embedding 2, or OLE2, the basic INI file format is too limiting for the types of reference data that needs to be stored, so most new Windows applications make use of the Registration Database or Registry to store a wide variety of information about applications. The Registry has been a part of Windows since v3.0, and it can be accessed by anyone who can locate the Windows Registry edit utility in the Windows directory.

This utility is made less obvious on purpose because it is very possible to corrupt the Registry if the common "trial and error" approach is used by inexperienced users (and by some experienced users too !). If the Registry becomes corrupted, you may not be able to reboot Windows, and a full re-install will often be necessary.

If you make a mistake and corrupt the Registry, providing you have previously saved the registry, you can restore the registry as follows.

- 1 Click the Start button, and then click Shut Down.
- 2 Click Restart The Computer In MS-DOS Mode, and then click Yes.
- 3 Change to your Windows directory. For example, if your Windows directory is C:\Windows, you would type the following:
cd c:\windows
- 4 Type the following commands, pressing ENTER after each one. (Note that System.da0 and User.da0 contain the number zero.)
attrib -h -r -s system.dat
attrib -h -r -s system.da0
copy system.da0 system.dat
attrib -h -r -s user.dat
attrib -h -r -s user.da0

copy user.da0 user.dat
- 5 Restart your computer.

Following this procedure will restore your registry to its state when you last successfully started your computer.

The Registry contains information that is critical to the correct operation of your computer, so before accessing and modifying the contents of the Registry, you should make a back up copy of the Registry. You can do this using the **RegEdt32** (Win NT) or **RegEdit** (Win 95, also found in NT) utility, as detailed in the REGEDIT on-line help.

On-line Help information on the Registry is very limited in Windows 95, so we will discuss it in moderate detail here. A full discussion of the Registry data formats and structures is beyond the scope of this User Manual, so if you need more detailed information, may we suggest a visit to your nearest computer bookstore to look in the Windows programming section.

The Registry is a hierarchical database made up of keys that are linked together to form hierarchies or tree structures similar in general format to the file system you are likely familiar with by now. These keys are the fundamental entities in the database.

The Registry has six keys, also called **root keys**, that serve as entrypoints to the database for any application. **Links** provide a mechanism to traverse the database from a root key to other subkeys. The link between two keys also serves to establish the relationship of a subkey, so the subkey is further from the root of the hierarchy than the other key in the link.

Each key has a **name** and a default **value**. A key can also have other named values associated with it. A value can be named or unnamed and has its own storage area for a data value. The data value can store binary, numerical, string, delimited strings,

or other types of data.

PIXCL provides six predefined constants for Registry access to permanently open keys.

@RDB_CLASSES_ROOT

This tree stores information about file type associations and most applications parameters. Newly installed applications will generally store information in new keys created in this tree. A PIXCL application is most likely to access registry data in this tree.

@RDB_CURRENT_USER

Application events, configuration, system and software information.

@RDB_LOCAL_MACHINE

Application events, configuration, system and software information.

@RDB_USERS

Information about user accounts and individual software and setup options.

@RDB_CURRENT_CONFIG

Display and system parameters.

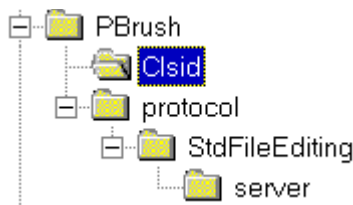
@RDB_DYN_DATA

Configuration Manager and Performance statistics.

Handles are usually negative integers. You can access a permanently open handle (i.e. one of the predefined constants), or by specifying a handle returned by the [RDBOpenKey](#) command.

Using the Registry access commands in PIXCL

To use the Registry access commands in PiXCL, you will need to become familiar with using the **RegEdit** Registry Editor, because in most cases what you will be wanting to do is extract and possibly update registry entries from within a PiXCL application program. For example, many programs keep a list of the 'n' previously accessed files, or perhaps the current file types, working and storage directories.



If you start up RegEdit, and look in the CLASSES_ROOT tree, and find the entry for Pbrush, you will see the subtree shown left with some common subkey names.

| | |
|-----------------|---|
| Clsid | Class Identifier, used with OLE 2. This is obtained by developers from Microsoft and is supposed to be unique for any applications. |
| protocol | Also used with OLE 2 |
| shell | The command used with the command shell |

Other commonly used subkey names.

| | |
|-----------------------|---|
| DefaultIcon | The icon that is used in the Start bar e.g. app.exe,1 |
| RecentFileList | A delimited list of 'n' files previously edited. |

Subkey names are arbitrary, and can contain spaces if required. For example, **RecentFileList** and **Recent File List** (with spaces) are different subkeys.

PIXCL Commands to access the Registry

There are eight commands in PiXCL 4.4 to read and write the Registry. Briefly, these are

[RDBCcloseKey\(InHandle, Result\)](#)

The **RDBCcloseKey** function releases the handle of the specified key. Permanently open keys cannot be closed, and if used, Result returns 0.

[RDBCcreateKey\(InHandle, SubKey\\$, ObjectType\\$, OutHandle, Result\)](#)

The **RDBCcreateKey** function creates the specified key. If the key already exists in the registry, the function opens it.

[RDBDeleteKey\(InHandle, SubKey\\$, Result\)](#)

Windows 95: The **RDBDeleteKey** function deletes a key and all its descendants.

Windows NT: The **RDBDeleteKey** function deletes the specified key. This function cannot delete a key that has subkeys, so it is necessary to delete the last key in each branch at a time.

[RDBEnumKey\(InHandle, Index, SubKeyName\\$, ClassName\\$, Result\)](#)

RDBEnumKey enumerates subkeys of the specified open registry key [InHandle](#). The function retrieves information about one subkey each time it is called.

[RDBOpenKey\(InHandle, SubKey\\$, OutHandle\)](#)

RDBOpenKey returns a handle of the specified subkey.

[RDBQueryKey\(InHandle, ClassName\\$, NumberOfSubKeys, NumberOfValues, Result\)](#)

RDBQueryKey returns an assortment of information about a specified key or predefined constant, useful in defining other registry commands.

[RDBQueryValue\(InHandle, SubKey\\$, SubKeyRtn\\$, Result\)](#)

RDBQueryValue returns the value associated with the specified subkey.

[RDBSetValue\(InHandle, SubKey\\$, Value\\$, TOKEN, Result\)](#)

The **RDBSetValue** function stores data in the value field of an open registry key. It can also set additional value and type information for the specified key.

For more detailed information, see

[RDBCcloseKey](#), [RDBCcreateKey](#), [RDBDeleteKey](#), [RDBEnumKey](#), [RDBOpenKey](#), [RDBQueryKey](#), [RDBQueryValue](#), [RDBSetValue](#)

Building Runtime .EXE Files

Both FreePiXCL 4.48 and PiXCL Registered versions come with a free unlimited runtime .EXE builder that you can use to share your programs with as many people as you want—even if they don't own PiXCL. Here are the main features of the runtime:

- Your PiXCL program is transformed into a standalone executable (.EXE) file.
- You can start the resulting .EXE file just like any other Windows program. Command line arguments are supported.
- No PiXCL interpreter is needed as it is built into the EXE.
- You can distribute your FreePiXCL and PiXCL programs royalty free.

Runtime Builder Directions

To turn a PiXCL program into a standalone executable (.EXE) file, you use PXL_make50.EXE, which is part of the PiXCL MDI Editor, and is invoked by clicking the appropriate button. It prompts you for the following items:

- The name of your PiXCL script file (for example, DEMO.PXL).
- The name you want to give the final executable file (for example, DEMO.EXE).
- Optionally, you can specify several Window style variations to the Runtime window style.

PXL_make will then create the final executable file on disk. For more information, see the PiXCL MDI Editor Help and PXL_make Help.

Notes:

The PiXCL Runtime does not let you combine multiple PiXCL scripts into a single .EXE file. Instead, you must make a separate .EXE file for each script. You will most likely find that this is never an issue, since a PiXCL script can be up to 1GB in length.

Creating your own CD-ROMs

Windows 95 / 98 / NT / 2000 support a feature called AutoPlay that enables a CD-ROM title to automatically start when the CD-ROM is inserted into the drive. The AutoPlay feature can also be turned on and off by setting a value in the Control Panel, and rebooting Windows so that the change takes effect.

AutoPlay automates the procedures for installing and configuring products designed for Windows-based platforms that are distributed on compact discs. When you insert a disc containing AutoPlay into a CD-ROM drive on a computer running Windows, AutoPlay automatically starts an application on the disc that installs, configures, and runs the selected product.

Enabling AutoPlay on a CD-ROM

If you are building an AutoPlay CD-ROM title, you must create an [Autorun.inf](#) file in the root directory of your CD application.

Suppressing AutoPlay

You can manually prevent the [Autorun.inf](#) file on a compact disc from being parsed and carried out by holding down the SHIFT key when you insert the CD-ROM disc.

The Autorun.inf File

The [Autorun.inf](#) file is a text file located in the root directory of the CD-ROM disc. This file contains the name of the startup application on the disc (the application that runs automatically when the disc is inserted in the CD-ROM drive), and the icon that you want to represent the AutoPlay-enabled compact disc in the Windows user interface. The [Autorun.inf](#) file also can contain optional menu commands that you want added to the shortcut menu, which is displayed when the user right-clicks the CD-ROM icon.

At a minimum, an [Autorun.inf](#) file contains three lines of text and identifies the startup application and the icon, as shown in the following example:

```
[autorun]
open=filename.exe
icon=filename.ico
```

An application's setup program is often called Setup.exe, but it can be any name you choose.

The `[autorun]` section identifies the lines that follow it as AutoPlay commands. An `[autorun]` section is required in every [Autorun.inf](#) file. The open command specifies the path and file name of the startup application, and the icon command specifies the file that contains the icon information.

The open command also supports arguments and path statements. For example, you may want a program in a sub-directory to be started by AutoPlay, because the root directory has other information in it. The open command might be in this case ...

```
open=.\subdir\filename.exe arg#1
```

The icon command also has several forms. You can specify an icon file as in the example above, and write the icon file into the root directory of your CD, or alternatively, specify an icon resource in an executable file. For example, you could use

```
open=setup.exe
icon=setup.exe,1
```

The “,1” indicates that the first icon in the Setup.exe should be used as the default icon that appears in the Windows TaskBar when the application specified by the open command is running.

You will want to test that the AutoPlay function on your new CD title is working correctly, without wasting a blank CD. The solution is to enable AutoPlay on any disk. This is done by changing the [NoDriveTypeAutoRun](#) Value

Setting the NoDriveTypeAutoRun Value

The **NoDriveTypeAutoRun** value in the registry is a 4-byte binary data value of the type REG_BINARY. The first byte of this value represents different kinds of drives that can be excluded from working with AutoPlay. The initial setting for this byte is 0x95, which excludes the unrecognized type drive, DRIVE_UNKNOWN, DRIVE_REMOVABLE, and DRIVE_FIXED media types from being used with AutoPlay.

You can enable a floppy disk drive or other removable disk drive (e.g. read-write optical drives, Iomega ZIP™ and JAZZ™ drives) for AutoPlay by resetting bit 2 to zero, or by specifying the value 0x91 to maintain the rest of the initial settings. A table identifying the bits, bitmask constants, and a brief description of the drives follows:

| Bit number | Bitmask constant | Description |
|--------------------|--------------------------|--|
| 0 (low-order bit) | DRIVE_UNKNOWN | Drive type not identified. |
| 1 | DRIVE_NO_ROOT_DIR | Root directory does not exist. |
| 2 | DRIVE_REMOVABLE | Disk can be removed from drive. |
| 3 | DRIVE_FIXED | Disk cannot be removed from drive (a hard disk). |
| 4 | DRIVE_REMOTE | Network drive. |
| 5 | DRIVE_CDROM | CD-ROM drive. |
| 6 | DRIVE_RAMDISK | RAM disk. |
| 7 (high-order bit) | Reserved for future use. | |

Note For Windows 95 and NT, you must restart Windows Explorer before any changes take effect.

Using the Registry to Change AutoPlay Settings

The registry is a feature of Windows that supersedes the initialization (.INI) and application configuration files. For information about manipulating the registry within a PiXCL application, see [Using the Registry](#).

If your product records and uses initialization information, you can use the registry to store and retrieve this information. Your startup application can use the information in the registry to determine whether the product needs to be installed. If there are no registry entries for your product—which means your product is being used for the first time—you could display a dialog box that lists the setup options. If your product is listed in the registry—which means it has already been installed—you could skip the setup options.

By changing the system registry, you can enable a computer to read the [Autorun.inf](#) file from a floppy or other removable disk. This feature of implementing AutoPlay on a floppy disk is provided only to help you debug your [Autorun.inf](#) files before you burn the compact disc. AutoPlay is intended for public distribution on compact disc only. To implement AutoPlay on a floppy disk, carry out the following procedure:

- In the Registry Editor (Regedit.exe), click Edit, and then click Find.
- In the Find What box, type the following, and then click Find Next:
NoDriveTypeAutoRun
- Click Edit, and then click Modify.
- Change the data of the **NoDriveTypeAutoRun** value from 0000 95 00 00 00 to 0000 91 00 00 00, and then click OK.

This enables AutoPlay on any drive. You must, however, start AutoPlay manually when it is installed on a removable disk. To do this, double-click the floppy disk icon, or right-click the floppy disk icon, and then click AutoPlay.

- After you complete your tests of [Autorun.inf](#), reset the value of **NoDriveTypeAutoRun** to 0000 95 00 00 00.

Note #1: Because implementing AutoPlay on a floppy disk provides an easy way to spread computer viruses, it is appropriate to suspect that any publicly distributed floppy disk that contains [Autorun.inf](#) files is contaminated.

Note #2: It is considered polite practice for a CD-ROM title to be able to run directly from the CD-ROM, without installed any files to the user's hard disk. A PiXCL application is self-contained, and so long as PXL_msg.DLL, PXLimage.DLL and PXLbtmps.DLL are present in the same directory as the EXE file, your application will run from the CD-ROM.

GeoPiXCL: add PXLgeofn.DLL, PXLgeofmt.DLL, PXLshape.DLL and any user DLLs into the same directory as the PXLimage.DLL.

DrawArc(x1 ,y1, x2, y2 ,x3, y3, x4, y4)

Draws an elliptical arc. The border is drawn according to the default pen, or the pen last set with the UsePen command.

DrawBackGround

Draw the background in either the default color or the color last set in the UseBackGround command. Backgrounds can also be drawn as a bitmap with the DrawBitMap and DrawSizedBitMap commands.

DrawBitMap(x, y, FileName\$)

Draw the bitmap specified, starting at the top left corner. Bitmaps must have either a .BMP, .RLE, .RAS, .TIF, .JPG, .JIF, .PCD, PNG, .PCX or .TGA extension. Bitmaps will be drawn according to the available number of colors in the Windows palette.

The DrawPreviewBitmap command has the same syntax.

DrawZoomedBitmap(x1,y1,x2,y2,FileName\$,px,py,ZoomFactor)

Zoom the specified bitmap into the target rectangle, on pixel px,py, at ZoomFactor 1 to 16.

DrawChord(x1, y1, x2, y2, x3, y3, x4, y4)

Draw a chord with the default pen or pen last defined in the UsePen command.

DrawEllipse(x1, y1, x2, y2)

Draw an ellipse or circle with the default or current pen defined by the UsePen command.

DrawFlood(x, y, r, g, b)

Fills an enclosed shape or area bounded with the specified color. The shape can be square or a random polygon. The fill color is either the default color or the color specified by the last UseBrush command.

DrawFloodExt(x,y,r,g,b,BORDER/SURFACE)

Fills out from a bounded enclosed shape or area while the specified color is found. The shape can be square or a random polygon. The fill color is either the default color or the color specified by the last UseBrush command.

DrawLine(x1, y1, x2, y2)

Draw a line between the two coordinate points with the pen defined in the last UsePen command.

DrawIcon(x, y, width, height, TOKEN)

Draws one of the sixteen embedded PiXCL icons or one of four system icons at the specified coordinates. If either width or height are zero, the icon is drawn at the default size.

DrawIconFile(x, y, width, height, File\$,TOKEN1,TOKEN2)

Draws an external icon or cursor file at the specified location and size. TOKEN1 defines CURSOR or ICON, TOKEN2 defines OPAQUE or TRANSPARENT.

DrawNumber(x, y, n)

Draw the number at the specified coordinates, in the default font or the font specified in the last UseFont command.

DrawPie(x1, y1, x2, y2, x3, y3, x4, y4)

Draws a pie wedge.

SetColorPalette(BITMAP/GENERATE)

Controls whether PiXCL uses a BitMap's own color palette or generates its own evenly distributed color palette.

UseBrush(token, r, g, b)

Fill an enclosed region with the color and TOKEN pattern. Tokens are

SOLID

DIAGONALUP

DIAGONALDOWN

DIAGONALCROSS

HORIZONTAL

VERTICAL

CROSS

NULL

UseFont(FontName\$,Width,Height,tokens,r,g,b)

Draw text in the specified font, size and color. Three Tokens are required, one each from each pair listed below.

BOLD / NOBOLD

ITALIC / NOITALIC

UNDERLINE / NOUNDERLINE

UsePen(token,Width,r,g,b)

Use the specified width and color pen in DrawLine commands. Token is

| | |
|------------|-------------------------------|
| SOLID | draw a solid line _____ |
| NULL | no line is visible |
| DASH | draw a dashed line - - - - - |
| DOT | draw a dotted line |
| DASHDOTDOT | draw a line _ . . _ . . _ . . |

DrawEdgeRectangle(x1,y1,x2,y2,TOKEN1,TOKEN2,TOKEN3)

Draw a rectangle at the coordinates above in the current background color, with the selected 3D border styles.

DrawRectangle(x1, y1, x2, y2)

Draw a rectangle at the coordinates above. The border is in the current [UsePen](#) command, and the rectangle is filled with the current [UseBrush](#) color, if its TOKEN is not to NULL.

DrawShadeRectangle(x1,y1,x2,y2,r1,g1,b1,r2,g2,b2,TOKEN)

Draw a color gradient rectangle at the coordinates above. either TOPBOTTOM or BOTTOMTOP, from RGB color 1 to RGB color 2.

DrawRoundRectangle(x1, y1, x2, y2, x3, y3)

Draws a rectangle with rounded corners, radius x3, y3. The border is in the current [UsePen](#) command, and the rectangle is filled with the current [UseBrush](#) color, if its TOKEN is not to NULL.

DrawSizedBitMap(x1, y1, x2, y2, FileName\$)

Draw the named Bitmap file (.BMP or .RLE only) into the rectangle specified. RLE bitmaps should be created with multiples of four pixels, or else the fill pixels will appear as a black vertical line on the side of the image. BMP files of the same size do not have this problem. Bitmaps can be 1, 4, 8 or 24 bits per pixel. The image will be displayed in the available colors in the Windows video driver static palette. In a 256 color driver the static map is 20 colors, and in 64K color driver, the static map is usually 4096 colors.

DrawStatusText(x1,y1,x2,y2,Text\$,NOBORDER | POPOUT

Draws a single line of text in a status bar with a medium gray background (192,192,192) and either a 3D border (the default), or no border, according to the TOKEN.

DrawStatusWinText(Part,Text\$)

Draws the relevant text into the specified part of the status bar, if it has been enabled.

DrawText(x,y,Text\$)

Draws the specified single line text string in the current font at the coordinates.

DrawTextExt(x1, y1, x2, y2,Text\$,LEFT | CENTER | RIGHT)

Draws a single or multi-line text string in the current font, in the rectangle specified, and justified according to the TOKEN argument.

GetBitMapDim(Filename\$,Lines,Pixels,BitsPerPixel)

Access the specified bitmap file and report the number of lines, pixels per line and the number of bits per pixel (1, 4, 8 or 24).

GetScreenCaps(TOKEN, Result)

Available TOKENS are

HORZSIZE / VERTSIZE get the horizontal or vertical size in millimeters (METRIC mode)
HORZRES / VERTRES get the horizontal or vertical size in pixels (PIXEL mode)
NUMCOLORS get the number of colors in the static color map.

For 8 bit displays, this returns a number (typically 20), and for displays with more than 256 colors, returns the value -1. For 24 bit displays, the static color map has 4096 entries.

DESKTOPHORZRES For NT only.
DESKTOPVERTRES get the virtual horizontal or vertical size in pixels (PIXEL mode)

NUMBRUSHES
NUMPENS
NUMFONTS
SIZEPALETTE
COLORRES
NUMRESERVED

UseBackGround(OPAQUE/TRANSPARENT,r,g,b)

Controls the background mode and color.

OPAQUE fills the background of character cells, and is faster.

TRANSPARENT does not fill the background of character cells, and is the slower mode.

UseCoordinates(PIXEL / METRIC)

Use the appropriate screen coordinate system. PIXEL mode uses a 1000x1000 grid and is the most accurate for drawing graphics and text. METRIC mode uses increments of 1mm, and is far less precise.

For applications that support many resolutions, it often looks better to use PIXEL mode for all displays.

DirChange(DirectoryName\$,Result)

Sets the current directory to the specified drive and path. It is often useful to check if the target directory actually exists.

DirGet(DirectoryName\$)

Gets the current directory, including the full disk and path. It is often used for tracking the directory of source files, libraries, or getting the new directory selected in a FileGet operation. The current directory can be changed by using the CHANGEDIR token in the FileGet command.

We suggest that DirGet() be used in the initialization of all PiXCL programs, so the start up directory (the initial current directory) can be saved in a string variable for later use.

DirGetSystem(DirectoryName\$)

Gets the Windows system directory and path. This will usually be "c:\windows\system".

DirGetWindows(DirectoryName\$)

Gets the Windows directory. This will usually be "c:\windows".

DirMake(DirectoryName\$, Result)

Creates a new directory. If successful, Result is set to 1, otherwise it is set to 0.

DirRemove(DirectoryName\$, Result)

Removes an existing directory.

DiskChange(DriveLetter\$, Result)

Sets the active disk drive.

DropFileServer(ENABLE | DISABLE,FileList\$)

Enables or disables the dropfile server function. [FileList\\$](#) should be a null string with the [DISABLE](#) call.

FileCopy(SourceName\$,DestinationName\$,Result)

Copies a file from one directory to another. See also the FileMove command.

FileExtension(FileName\$,Extension\$,Result)

Extracts the extension of a disk-path-filename string.

FileName(FileName\$,RootName\$,Result)

Extracts the rootname of a disk-path-filename string.

FilePath(FileName\$,Path\$,Result)

Extracts the Path of a disk-path-filename string.

FileGet(Filter\$,InitFile\$,InitDir\$,Caption\$,token,ChosenFile\$)

Lets you get a file using the Windows common dialog box library COMMDLG.DLL. The available TOKENS are

| | |
|------------------------|--|
| CHANGEDIR | On exit of the command, change the current disk\directory to the selected directory from the initial directory specified. This will affect the result of a DirGet() command. |
| CHANGEDIRMULTI | Enables file multi-select operations. |
| CHANGEDIR_EXIST | Checks if the file and path exists. An error dialog appears if the file cannot be located. |
| CHANGEDIRMULTI_EXIST | Combination of the above. |
| NOCHANGEDIR | If a new disk\directory was selected, do not change the current directory setting. |
| NOCHANGEDIRMULTI | Enables file multi-select operations. |
| NOCHANGEDIR_EXIST | Checks if the file and path exists. An error dialog appears if the file cannot be located. |
| NOCHANGEDIRMULTI_EXIST | Combination of the above. |

FileDelete(FileName\$, Result)

Deletes the specified file or files. Wildcards "*" and "?" are allowed.

FileExist(Filename\$, Result)

Lets you determine if a file exists.

FileGetDate(Filename\$, Year, Month, Day, Result)

Reads in a file's directory entry.

FileGetDateExt(Filename\$, TOKEN, Year, Month, DayOfWeek, Day, Result)

Gets the extended information in Windows 95 and NT 3.51.

FileGetSize(Filename\$,Size)

Gets the size of a specified file in bytes.

FileGetTime(Filename\$,Hours, Minutes,Seconds,Result)

Reads in the file's time-stamp.

FileMove(SourceName\$,DestinationName\$,Result)

Moves the file from one directory to another.

FileRead_ASCII(FileName\$,Offset,Length,Field\$,Res)

Read a field from a text file.

FileRead_INI(INI_file\$,Section\$,Key\$,Return\$)

Reads the specified Windows or private initialization file section or key string.

FileRename(SourceName\$,DestinationName\$,Result)

Changes the name of one or more files.

FileWrite_ASCII(FileName\$,Offset,Length,Field\$,Result)

Writes a field to a text file.

FileWrite_INI(INI_file\$,Section\$,Key\$,String\$, (NO)WARN,Result)

Writes the specified Windows or private initialization file section or key string.

GetDiskSpace(Disk\$,Type\$,TotalSpace,FreeSpace)

Returns the disk type (FIXED, REMOVABLE, REMOTE, CD-ROM, RAMDISK, UNKNOWN), plus the total space and available free space in Kbytes.

GetVolumeType(RootDir\$,FileSysType\$,Result)

Gets the filesystem of the named root directory (FAT, NTFS, HPFS)

ClipboardAppend(String\$, Result)

Adds text to the end of the Windows Clipboard.

ClipboardEmpty

Empties the Windows Clipboard

ClipboardGet(String\$, Result)

Reads text from the Windows Clipboard.

ClipboardPut(String\$, Result)

Copies String\$ to the Windows Clipboard, replacing the current contents.

Help Topics for PIXCL v4.0

Click on the green text to navigate around the hypertext links in the help document. Additional help is available by pressing the F1 key.

The Windows GDI

The Graphics Display Interface is the set of high level commands that programs like PiXCL can call to draw pixels on the screen, in any format or style. The complexities of the specific hardware is insulated from the application by the Windows video driver itself.

Ansi(String\$, Code)

Returns the ANSI code for the first character in the string.

Chr(Code, String\$)

Returns a one-character string based on the specified ANSI code.

Instr(String1\$, String2\$, Location)

Finds the starting location of one string within another.

LCase(String\$)

Converts a string to lowercase.

Left(String\$, Place, Result\$)

Returns a specified number of characters from the left of the string.

LeftOf(String\$, Location, Result\$)

Returns all characters to the left of a location in a string.

Len(String\$, Length)

Returns the length of a string.

Pad(String\$, Length)

Pads a string with spaces to a specified length.

Right(String\$, Places, Result)

Returns a specified number of characters from the right of the string.

RightOf(String\$, Location, Result)

Returns all characters to the right of a location in a string.

Space(String\$, Length)

Initializes a string to a specified number of spaces.

Str(Number, String\$)

Converts a number to a string.

StrCmp(String1\$, String2\$, Result)

Compares two strings (case sensitive).

StrCmpl(String1\$, String2\$, Result)

Compares two strings (case insensitive).

StrRepl(String1\$, OldSubString\$, NewSubString\$, Result)

Replaces the first instance of a substring in a string.

Substr(String\$, Places, Location, Result\$)

Returns a specified number of characters from a string starting at a specified location.

Trim(String\$)

Trims trailing spaces from a string.

TrimExt(String\$,L|R|A)

Trims leading and/or trailing spaces from a string.

UCase(String\$)

Converts a string to upper case.

Val(String\$, Number, Result)

Converts a string to a number. If successful the number is in the range 0 - 65535.

ChangeMenuItem(Item\$, CHECK/UNCHECK/GRAY/ENABLE, Result)

Checks, unchecks, grays, or enables a pop-up menu item.

GetMenuStatus(Item\$, CHECKED/GRAYED, Result)

Checks whether a menu item is grayed or checked.

InfoMenu(REMOVE/ADD)

Lets you remove or replace the PiXCL Info menu item that when enabled, appears on the right hand side of the menu bar.

SetMenu(Top1\$, IGNORE/Label, ..., ENDPOPUP)

Builds a custom menu. The command [SetMenu\(\)](#) clears any existing menus, and will leave the menu bar blank.

Gosub <Label>

Executes a subroutine. A "Return" statement is required.

Goto <Label>

Branches to <Label>.

If <condition> then <commands>

Executes commands conditionally. If true, any additional commands on the SAME line will be executed.

If-Else-Endif

Executes commands conditionally, using the structured If.

```
If <condition>  
    commands  
Else  
    commands  
Endif
```

Return

Returns from a subroutine.

Button(Btn1_x1, btn1_y1, Btn1_x2, Btn1_y2, STYLE,Btn1Text\$, Label, . . .)

Creates custom 3-D command pushbuttons, radio buttons, checkboxes or group boxes in any combination. STYLE token can be PUSH|RADIO|AUTORADIO|CHECK|AUTOCHECK|GROUP. These are the current system color, usually gray. The command [Button\(\)](#) clears any existing buttons assignments.

[Full details.](#)

ListBox(Caption\$, List\$, Delimiter\$, Result\$)

Displays a dialog box with a list box inside.

ListBoxExt(Label\$,List\$,Delim\$,Help\$,Res\$)

Displays a dialog box with an extended multi-line list box inside, as well as optional Help messagebox

MessageBox(OK/..., DefaultButton, STOP/..., Text\$, Caption\$, ButtonPushed)

Creates a custom message box. The available button TOKENS are

OK OKCANCEL YESNO RETRYCANCEL YESNOCANCEL
ABORTRETRYIGNORE

The available icon TOKENS are

STOP INFORMATION EXCLAMATION QUESTION NOICON

SetKeyboard("a", label, "^a", label, vkey, label)

Sets up where the program branches to when the user presses a key.
The command [SetKeyBoard\(\)](#) clears any existing key assignments.

SetMenu(Item1\$, IGNORE/Label, ..., ENDPOPUP)

Creates a custom menu. There is a SEPARATOR token as well.
The command [SetMenu\(\)](#) clears any existing menus, and will leave the menu bar blank.

SetWaitMode(NULL/FOCUS)

Controls how PiXCL behaves after a Run command starts another application and PiXCL encounters a WaitInput command in your script.

TextBox(Text\$, Caption\$, Input\$, ButtonPushed)

Displays a dialog box with a single-line edit control.

TextBoxExt(Text\$,Label\$,Help\$,Input\$,Btn)

Displays a dialog box with a single-line edit control and optional Help messagebox.

UseCursor(TOKEN)

Controls the appearance of the mouse cursor. The available TOKENS are

APPSTARTING ARROW CROSS IBEAM
ICON NO SIZE SIZEALL SIZENESW
SIZENS SIZENWSE SIZEWE UPARROW
WAIT

EnumWindows(WindowList\$,VISIBLE/ALL,Delimiter\$)

Creates a delimited list of all parent windows.

SendKeys(WindowName\$,KeyStrokes\$,PauseRespond,PauseKeyStroke,Respond)

Sends virtual keystrokes to the Windows 95, NT 3.51 or Windows 3.1 application. Some restrictions apply. See details in text.

SetPriority(CommandLine\$,IDLE/NORMAL/HIGH,Result)

Sets the priority of the PiXCL process, or a process launched by PiXCL.

SetSendKeysPriority(TOKEN)

Available TOKENS are

LOWEST

BELOW_NORMAL

NORMAL

ABOVE_NORMAL

HIGHEST

SetWindow(MAXIMIZE/MINIMIZE/RESTORE)

Maximizes, minimizes, restores the PiXCL window.

StatusWindow(TOKEN_1,TOKEN_2,Parts,End1,End2,End3,End4)

ENABLE|DISABLE the status bar window at TOP|BOTTOM. See also the [DrawStatusWinText](#) command.

ProgressBar(ENABLE|DISABLE,x1,y1,x2,y2)

Enable or disable a progress bar at the specified co-ordinates, or if all co-ordinates are zero, at the bottom of the client area. Use the [UpdateProgressBar](#) command to modify the display.

UpdateProgressBar(Value,RELATIVE|ABSOLUTE|INCREMENT)

Update the the progress bar display according to the mode token. Use with the [ProgressBar](#) command.

UseCaption(Text\$)

Sets the text that appears in the title bar of the current or selected window.

WinClose(WindowName\$,Res)

Closes the specified window, if it exists. Use this command with the [EnumWindows\(...\)](#) command.

WinExist(Windowname\$, Result)

Determines whether an application is running. The exact name must be checked.

WinGetActive(Windowname\$)

Returns the name of the active application window. This is not usable with child windows.

WinGetLocation(WindowName\$,X1,Y1,X2,Y2,Res)

Get the screen position of the specified window. The exact name must be known or acquired with the [EnumWindows\(...\)](#) command. The returned co-ordinates can be negative.

WinLocate(Windowname\$, x1, y1, x2, y2, Result)

Locates a window at the specified co-ordinates. The co-ordinate is specified in the current mode, either METRIC or PIXEL. Negative co-ordinates are supported.

In PiXCL 4.0 and later, [WinLocate](#) or [WinShow](#) should be used in the program initialization stages to locate the PiXCL program window at the desired co-ordinates.

WinSetActive(Windowname\$, Result)

Activates the window whose title bar is specified by 'Windowname\$'.

WinShow(Windowname\$, TOKEN, Result)

Available TOKENs are

HIDE / UNHIDE / MINIMIZE / MAXIMIZE / RESTORE / TOPMOST / NOTOPMOST / TOP / BOTTOM / SHOMINNOACTIVE /
SHOWNOACTIVATE

Hides, unhides, maximizes, minimizes, or restores the specified window. In PiXCL 4.0 and later, WinLocate or WinShow should be used in the program initialization stages to locate the PiXCL program window at the desired co-ordinates.

AbortShutDown

Cancels the Windows NT shutdown process initiated by the Shutdown command.

Not supported under Windows 95.

WinTitle(Windowname\$, Title\$)

Sets the text that appears in the title bar of a window. This can be the current Window or any other primary Window that is open. It is not possible to change the title of child windows.

AboutPIXCL

Displays the standard PiXCL About dialog box. This is the same box that the [InfoMenu](#) command enables or removes.

AboutUser(Title\$, Box1String\$, Box2String\$)

Displays a customisable About box that can be used to describe user Runtime programs.

Beep

Sounds the bell. Putting a `WaitInput(150)` between Beep commands will make a suitable interval.

End

Terminates an PiXCL program. Windows will pass control to the next available Active Window. This will often but not necessarily be the Program Manager.

ExitWindows

Works with Windows NT only. Ends the Windows session. Windows will shut down.

FreeBitMap(fileName\$)

Frees the memory taken by the image the DrawBitmap command processes.

FreeBitMapAll

Removes all bitmaps from memory and recovers the memory space.

FreeVar(String\$)

Removes a string variable from the string variable list and recovers the memory it occupied.

FreeVarAll

Removes all string variables from the string variable list and reclaims the memory they occupied.

GetPixel(X,Y,r,g,b,Result)

Returns the red / green / blue pixel value at the designated co-ordinate in the PiXCL window client area.

GetCmdLine(CommandLine\$)

Returns the complete command line and any arguments used to start the current application. You can use the string commands to extract the necessary arguments.

GetScreenCaps(HORSIZE/VERTSIZE/HORZREZ/VERTREZ/NUMCOLORS, Result)

Returns information about the screen driver's capabilities - for example, the number of supported colors and resolution.

Logoff

Logs off Windows NT or Windows 95 by closing running programs including Program Manager or Explorer.

MessageBeep(TOKEN)

Plays the waveform sound associated with an entry in the [sounds] section of WIN.INI. The available TOKENS are

BEEP ASTERISK EXCLAMATION HAND QUESTION OK

WAVPlaySound(Sound\$, TOKEN, ALIAS/FILENAME,Result)

Plays a specified waveform audio sound or an entry in the [sounds] section of the registry. The available TOKENS are

SYNC ASYNC LOOP NOSTOP

Random(Range,RandomNumber)

Returns a random positive integer within the range specified.

Negate(Number)

Returns the arithmetic negation (two's complement) of the input number.

Set Variable = ..., **Set Variable\$ = ...**

Assigns an integer or string variable. The Set keyword is provided for compatibility with earlier versions of PiXCL.

WinVersion(Major, Minor, Build, Pack\$)

Provides the current Windows version and build numbers. In Windows NT these are reported as **3 51**, and in Windows 95 as **4 0**. The build number will vary for Windows NT depending on the processor type and release date. For Windows 95, the build number is reported as 950. Pack\$ is the Service Pack string in NT, or an arbitrary or null string in Win95/98.

Run("Transfer <ascii_filename> W/R")

Using the Clipboard, either Write or Read an ASCII file. The string passed to the Clipboard must be decoded for Read operations, and for Write operations overwrites the existing file. In PIXCL v4.0, the ascii transfer buffer size is 5 KB.

Run("RpBMPDim <bmp_filename>")

OBSOLETE v2.5 Extension Function: See the GetBitMapDim command.

This function reports the number of bits per pixel, and the number of pixels and lines in the specified BMP file, by passing and ASCII string to the Clipboard. The PIXCL script must get the string from the Clipboard and decode the desired information.

Shutdown(CpuName\$,Msg\$,Timeout,RESTART/NORESTART)

Shuts down Windows NT as though you had selected File Shutdown from the Program Manager.

Not supported under Windows 95.

WaitInput(), WaitInput(milliseconds)

Pauses a program a specified number of milliseconds, or indefinitely waits for user input.

WinHelp(HelpFile\$,COMMAND_TOKEN,KeyWord\$)

Activate the Windows Help sub-system.

DrawFrameControl(x1,y1,x2,y2,TYPE,STATE1,STATE2,Result)

FrameControls are the little bitmaps that are used to create controls in a window frame such as title bar buttons, scrollbar buttons, grips, as well as radio buttons and push buttons. The DrawFrameControl function draws a frame control of the specified type and style.

DrawCaption(Window\$,x1,y1,x2,y2,(NO)ICON,COLOR|SMALLCAP,(NO)INBUTTON,Result)

Any window caption and its icon can be drawn in the client area, even if the window is not visible. The background of the rectangle is the same as the current title bar background system color. The DrawCaption command is used to draw application buttons similar to those in the Windows task bar.

DrawAnimatedRects(WindowName\$,fx1,fy1,fx2,fy2, tx1,ty1,tx2,ty2,OPEN|CLOSE|CAPTION,Result)

The DrawAnimatedRects function draws a wire-frame rectangle and animates it to indicate the opening of an icon or the minimizing or maximizing of a window.

SetDrawMouse(FOREGND|BACKGND|BOTH|DISABLE)

Enables or disables drawing with the mouse using the current pen.

DrawTriangle(x1,y1,x2,y2,x3,y3)

Draws a triangle with the current pen and brush.

DrawPolygon(x1,y1,x2,y2,...,xn,yn)

Draws a convex or concave polygon with the current pen and brush.

DrawFocusRectangle(x1,y1,x2,y2)

Draws a rectangle with a dotted outline.

WAVSetPitch(Device,Pitch)

Sets the current pitch. Not supported on all cards.

WAVSetPlayRate(Device,PlayRate)

Sets the current playback rate. Not supported on all cards.

WAVSetVolume(Device, LVol, RVol,Result)

Sets the playback volume.

WAVGetDevCaps(Device,TOKEN,Return\$,Result)

Get a set of device parameters.

WAVGetNumDevs(Number)

Returns the number of sound devices loaded.

WAVGetPitch(Device,Pitch)

Gets the current pitch. Not supported on all cards.

WAVGetPlayRate(Device,PlayRate)

Gets the current playback rate. Not supported on all cards.

WAVGetVolume

Gets the current volume setting.

SetROPcode(TOKEN)

Sets the Raster Operation code. This is an advanced user command. See [SetROPcode](#).

For-Next Loops

```
For variable = n|variable To m|variable [By p|variable]
  commands
  If <condition> Then Break {optional}
Next
```

While Loops

loopvariable = value

While loopvariable=number|string

 commands

 If <condition> Then Break {optional}

EndWhile

StrRev(String\$)

Reverses the character order of a string.

InvertRectangle(x1,y1,x2,y2)

Inverts the colors of the specified rectangle. Windows will do its best to display the new color.

PXLResume(WindowsName\$,Res)

Sends a 'resume' message to the target PiXCL application. Has no effect on other programs.

RDBCcloseKey(InHandle, Result)

The **RDBCcloseKey** function releases the handle of the specified key. Permanently open keys cannot be closed, and if used, Result returns 0.

RDBCreateKey(InHandle, SubKey\$, ObjectType\$, OutHandle, Result)

The **RDBCreateKey** function creates the specified key. If the key already exists in the registry, the function opens it.

RDBDeleteKey(InHandle, SubKey\$,Result)

Windows 95: The **RDBDeleteKey** function deletes a key and all its descendents.

Windows NT: The **RDBDeleteKey** function deletes the specified key. This function cannot delete a key that has subkeys, so it is necessary to delete the last key in each branch at a time.

RDBEnumKey(InHandle,Index,SubKeyName\$,ClassName\$,Result)

RDBEnumKey enumerates subkeys of the specified open registry key [InHandle](#). The function retrieves information about one subkey each time it is called.

RDBOpenKey(InHandle,SubKey\$,OutHandle)

RDBOpenKey returns a handle of the specified subkey.

RDBQueryKey(InHandle,ClassName\$,NumberOfSubKeys,NumberOfValues,Result)

RDBQueryKey returns an assortment of information about a specified key or predefined constant, useful in defining other registry commands.

RDBQueryValue(InHandle,SubKey\$,SubKeyRtn\$,Result)

RDBQueryValue returns the value associated with the specified subkey.

RDBSetValue(InHandle,SubKey\$,Value\$,TOKEN, Result)

The **RDBSetValue** function stores data in the value field of an open registry key. It can also set additional value and type information for the specified key.

ChooseFont(Font\$,Width,Height,r,g,b,Bold,Italic,Underline,Strikeout)

Use a common dialog to select the current font and style.

ChooseColor(TOKEN,Red,Green,Blue)

Also **ChooseColor(TOKEN,Red,Green,Blue,X,Y,Title\$,Basic\$,Custom\$)**

Use a common dialog to choose a color for painting and drawing. Tokens are STD|SMALL|SMALLRGB|FULL.

CustomColor(r1,g1,b1,...r16,g16,b16)

Define the set of 16 custom colors available within the ChooseColor common dialog.

RotateRectangle(x1,y1,x2,y2,TOKEN,Rate,Repeat)

Rotates vertically or horizontally the defined rectangle in the client area.

FileSaveAs(Filter\$,InitFile\$,InitDir\$,Label\$,CHANGEDIR,Name\$)

Use a common dialog to select or create a save filename. This does not actually save the file. Essentially the same as the FileGet command. Tokens are the same as FileGet.

SetEditControl(x1,y1,x2,y2,TOKEN,Max,Min,Input\$,...)

Create any number of edit windows in the client area for text or numeric input. Tokens are STRING|NUMBER|NUMBERUD|PASSWORD. Generally used with a Button command to terminate input. Can have and UpDown control on numeric input with max and min values.

Edit strings are displayed in the current font.

SetPopupMenu(Item\$,Label,[SEPARATOR],Item\$... ENDPOPUP)

Create a floating menu invoked with the right mouse click in the client area.

PasswordBox(Title\$,Text\$,Btn1\$,Btn2\$,Btn,Password1\$)

Create a dialog with secure text entry.

ImageBox(Title\$,Image\$, Text\$,Btn1\$,Btn2\$,Btn)

Create a dialog with text and thumbnail image (all supported formats) display.

LoadDLL(DLLname\$,Result)

Loads a third party Dynamic Link Library.

FreeDLL(DLLname\$,Result)

Frees (i.e. unloads) a third party Dynamic Link Library.

SaveBitmap(ImageName\$,Result)

Saves the current bitmap in one of the supported bitmap formats.

SaveRectangle(x1,y1,x2,y2,ImageName\$,Result)

Saves the specified client area rectangle to an image file.

RotateRectangle(x1,y1,x2,y2,TOKEN,Rate,Count)

Rotates the specified client area rectangle at Rate, Count times.

GetSystemTime(Year,Month,DayofWeek,Day,Hour,Minute)

Gets the current system time (GMT).

SetSystemTime(Year,Month,DayofWeek,Day,Hour,Minute)

Sets the current system time (GMT).

GetLocalTime(Year,Month,DayofWeek,Day,Hour,Minute)

Gets the current local time (GMT +/- bias time).

SetLocalTime(Year,Month,DayofWeek,Day,Hour,Minute)

Sets the current local time (GMT +/- bias time).

GetTimeZone(Zone\$)

Gets the current timezone string.

TimeToASCII(SYSTEM|LOCAL,mode_TOKEN,Time\$)

Returns the time string in one of five format variations.

LoadBitmap(ImageFile\$,PREVIEW|FULL)

Loads an image into memory without displaying it. This is equivalent to [DrawSizedBitmap\(0,0,0,0,ImageFile\\$\)](#).

NumToHex(Number,Hex\$)

Converts a 32 bit number to the equivalent hexadecimal string.

HexToNum(Hex\$,Number,Result)

Converts an 8 character hexadecimal string into the equivalent number.
If the conversion fails, both Number and Result return 0.

GetSystemMetrics(TOKEN,Value)

Returns a value that defines a metric for a system window object, such as menubar size. Can also return bootmode and number of mouse buttons.

GetSysPowerStatus(AC,BFlag,BLifePc,BLife,LifeTime)

Returns a set of values for a DC power management system. Only relevant to laptop systems, or systems with built-in DC power supplies.

ComboBox(x1,y1,x2,y2,STYLE,List\$,Delim\$,Input\$)

Creates one or more ComboBox controls in SIMPLE, DROPDOWN, and DROPDOWNLIST styles.

FindExecutable(File\$,Path\$,EXE_File\$,Result)

Locate the executable file that is associated with the selected specific file. If no EXE is associated, EXE_File\$ returns NULL.

ReportMouse(x1,y1,x2,y2,xOff,yOff,xZ,yZ,TOKEN)

Report client area mouse coordinates and optionally RGB values in the status bar. Tokens are DISABLE, NORGB, RGB.

PXLresumeAt(ToWindow\$,LABEL,FromWindow\$,Res)

Send a message to another PiXCL application in a WaitInput() loop to start processing at a specific label.

Toolbar(MODE,SIZE,Index,State,Style,Tip\$,Label,...)

Create a standard windows toolbar with up to 48 buttons.

ToolWindow(x1,y1,x2,y2,TYPE,MODE,SIZE,Index,State,Style,Tip\$,Label,...)

Create a series of CHILD or POPUP toolwindows.

GetToolBarBtnStatus(ToolBar\$,Index,STATE,Result)

Check that ToolBar\$ button of Index is STATE.

ChangeToolbarBtn(Toolbar\$,Index,STATE,Result)

Change Toolbar\$ button of Index to STATE.

EnumChildWindows(Parent\$,Child\$,VISIBLE|ALL,Delimiter\$)

List all the child windows of the specified parent.

CustomizeToolBtn(ToolWindow\$)

Start the system customize toolbar or toolwindow button dialog. Set ToolWindow\$ to a NULL string to customize the toolbar.

ListLoadedBitmaps(List\$,Delimiter\$,Count)

Returns a delimited list of the images currently in the PiXCL Bitmap List. If none present, returns a null string and 0 count.

CountBitmapColors(Image\$,Count)

Returns the number of unique colors in any of the supported bitmap formats. Count returns 0 if the image is not found in memory or on disk, otherwise returns a number in the range 1 to a number \leq maximum number of colors defined by the image format, and not greater than the number of pixels in the image.

GetBackground(Red,Green,Blue)

Returns the current RGB background color.

DrawTrBitmap(x1,y1,Image\$,Tr,Tg,Tb)

Draws the bitmap with transparent color Tr,Tg,Tb.

DrawTrSizedBitmap(x1,y1,x2,y2,Image\$,Tr,Tg,Tb)

Draws the sized bitmap with transparent color Tr,Tg,Tb.

SetDrawMode(FOREGND|BACKGND|BOTH)

Sets the mode for the DrawBitmap commands. Often used in simple animation.

DrawBackgroundRegion(x1,x2,y1,y2)

Copies the region from the background memory to the client area bitmap. Often used in simple animation.

PrintBitmap(Image\$,SETUP|PRINT,Result)

Prints the selected bitmap to the current printer identified by the SETUP token pass.

PrintFile(Filename\$,Result)

Prints a document file with the application associated with the file type. The application must be installed and available.

WinAdjustRect(x1,y1,x2,y2,(NO)MENU,wx1,wy1,wx2,wy2)

Returns the necessary window coordinates for the desired client area.

DrawPolyLine(x1,y1,...,xn,yn)

Draw a connected series of line segments, using the current pen.

DrawPolyCurve(x1,y1,...,xn,yn)

Draw a cubic Bezier curve using the specified points, using the current pen.

ClearCommPort(COMx)

Clears the buffers and resets error status on the specified port.

GetCommPort(COMx,baud,data,parity,stop,X)

Gets the selected port current settings.

ReadCommPort(COMx,Data\$)

Reads the selected port data into a string variable.

SetCommPort(COMx,Settings\$,XON|XOFF,Res)

Sets the selected port parameters eg "9600,8,N,1"

WaitCommEvent(R|W,COMx,<label>,Timeout)

Waits for a comm port read or write event. Follow with a WaitInput().

WriteCommPort(COMx,Data\$)

Write the string variable to the selected port.

EscCommFunction(COMx,token)

Send one of CLRDTR, CLRRTS, SETDTR, SETRTS, SETXOFF, SETXON, SETBREAK, CLRBREAK to the specified comm port.

UseBrushPattern(ImageName\$)

Set the current brush to a user defined 8x8 bitmap. If the pattern bitmap is not in the PiXCL image list, it is loaded.

DrawGrid(x1,y1,x2,y2,Hcell,Vcell,R,G,B,border_style)

Draw a grid of cell size in the rectangle, using a 1 pixel wide color pen. Border styles are NONE, STYLE_1, STYLE_2, STYLE_3.

DragAcceptFile(ENABLE|DISABLE,<label>)

Enables or disables drag-and-drop operations to <label>.

GetDragList(List\$)

Get the file list that was dropped into the PiXCL application window.

TWAIN_AcquireNative(Image\$,TOKEN,Handle)

Acquire an image and load it into the PiXCL image list.

TWAIN_AcquireToClipboard(Result)

Acquire an image and write it to the clipboard,

TWAIN_AcquireToFilename(Filename\$,Result)

Acquire the image and write it direct to the file on disk.

TWAIN_CloseSource(Result)

Close the current data source.

TWAIN_CloseSourceManager(Result)

Close the Source Manager i.e. TWAIN_32.DLL

TWAIN_DisableSource(Result)

Disable the current data source.

TWAIN_EnableSource(Result)

Enable the current data source.

TWAIN_GetBitDepth(Bits)

Return the bit depth (bits/color/channel) of the current source.

TWAIN_GetBitmapParams(Handle,Lines,Pixels,Bits,Colors)

Return the parameters of the image loaded into the PiXCL list from the TWAIN data source.

TWAIN_GetCurrentRes(Resolution)

Return the current scanner resolution.

TWAIN_GetCurrentUnits(Units)

Return the current source device unit setting.

TWAIN_GetPixelFormat(Pixel_Type)

Return the current type setting.

TWAIN_Getstate(Result)

Return the current data source state.

TWAIN_IsAvailable(Result)

Result = 0 if no TWAIN devices are known, otherwise = 1.

TWAIN_LoadSourceManager(Result)

Loads TWAIN_32.DLL data source manager.

TWAIN_OpenDefaultSource(Result)

Opens the current data source and displays its dialog, if present.

TWAIN_OpenSourceManager(Result)

Opens the source manager, TWAIN_32.DLL.

TWAIN_PxlVersion(Result)

Returns the version of PXLtwain32.DLL. Used for debug purposes. Result will be 106 or greater. i.e. v1.06 or later.

TWAIN_SelectSource(Result)

Displays the Select Source dialog from TWAIN_32.DLL.

TWAIN_SetBitDepth(Depth)

Tries to set the source device bit depth.

TWAIN_SetCurrentRes(Resolution)

Try to set the current resolution.

TWAIN_SetCurrentUnits(Units)

Set the current unit parameter.

TWAIN_SetPixelType(Type)

Try to set the current pixel type.

TWAIN_UnloadSourceManager(Result)

Unload TWAIN_32.DLL.

GetCopyDataMsg(Message\$)

Retrieves the ASCII string message that has been sent to the PiXCL application.

SendCopyDataMsg(Window\$,Message\$)

Send a label and message string to the named window.

FlashBMWindow(WinID,TOGGLE|RESET)

Change the state (active or inactive) of a bitmap window title bar.

SetBMW[Right]Mouse(WinID,Label,x,y,...)

Enables left or right mouse actions in bitmap windows.

DrawBMWPoint(WinID,x,y,STYLE)

Draws a point into the a bitmap and bitmap window at the coordinates, using the current pen color.

AutoProgressBar(ENABLE|DISABLE)

Enables (the default) or disables the progress bar display during image load and image processing operations.

GetScreenWorkArea(wx1,wy1,wx2,wy2)

Returns the coordinates of the screen area, less the area used by the Windows Taskbar/System tray, and any other tray-type windows.

DirExplore(DirName\$,Result)

Displays the contents of a directory in an Explorer window.

ShellAbout(Title\$,Text\$,ICON)

Displays the Windows Shell About dialog with user defined information.

DrawShadowText(x,y,Text\$,R,G,B,Offset)

Draw a text string with a defined shadow color, offset in the x, y directions.

DrawShadowTextExt(x1,y1,x2,y2,Text\$,LEFT|CENTER|RIGHT,R,G,B,Offset)

Draws a single or multi-line text string in the current font, in the rectangle specified, justified according to the TOKEN argument, and with a shadow color and x, y offset.

SetFontEscapement(AngleX10)

Set the escapement angle in 0.1 degrees that text is drawn using the DrawText family of commands. Default value is 0.

GetListBitMapDim(Filename\$,Lines,Pixels,BitsPerPixel)

Access the specified bitmap file in the PiXCL image list and report the number of lines, pixels per line and the number of bits per pixel.

DrawShadowNumber(x,y,Number,R,G,B,Offset)

Draw a number with a defined shadow color, offset in the x, y directions.

FileRead_Binary(File\$,Offset,Value,FWD|REV,Result)

Read an integer value from the file at the specified byte offset.

FileWrite_Binary(File\$,Offset,Value,FWD|REV,Result)

Write a 32-bit binary value at the specified offset in the file.

AppWindowHandle(Handle,Handle\$)

Return the binary and string version of the application window handle. This command will be used by programmers of extension functions.

FileGetTempName(Dir\$,Prefix\$,Number,File\$)

Generate a temporary filename.

GetTempPath(Path\$)

Return the current temporary file path from the process environment.

GetFontFace(Face\$)

Get the current font face for text writing.

GetEnvString(Delimiter\$,EnvStr\$)

Returns the current process environment string variables.

GetEnvVariable(Var\$,Value\$)

Returns an environment variable.

SetEnvVariable(Var\$,Value\$)

Sets a new value, or modifies an existing environment variable value.

StrReplAll(String1\$, OldSubString\$, NewSubString\$, Result)

Replaces all instances of a substring in a string.

ReadBitmapRect(Image\$,x1,y1,x2,y2,Result)

Read a rectangle of interest from a BMP or TIF file.

WriteBitmapRect(Image\$,x1,y1,x2,y2,Result)

Write a rectangle in a BMP or TIF file.

RemapImage(RGB_Array\$, Result)

Using a palette string array read from an ascii file, remap colors in the current image.

DrawFpNumber(x,y,Number&,Digits)

Draw a floating point number at the coordinates,with a defined number of significant digits.

DrawShadowFpNumber(x,y,Number&,Digits,R,G,B,Offset)

Draw a floating point number at the coordinates,with a defined number of significant digits, using the specified shadow color.

FpStr(Number&,Number\$)

Convert a floating point number into a string.

FpVal(Number\$,Number&,Result)

Convert a string into a floating point number.

ItemCount(List\$,Delimiter\$,Count)

Count the number of items in a string list.

ItemExtract(List\$,Delimiter\$,Index,Item\$,Result)

Extract a specific item from a string list.

ItemLocate(List\$,Delimiter\$,Item\$,Index)

Locate the index of an item, if it exists, in a string list.

ItemInsert(List\$,Delimiter\$,Index,Item\$,Result)

Insert, if possible, an item into a string list.

ItemRemove(List\$,Delimiter\$,Item,Result)

Remove, if possible, a specific item from a string list.

ItemSort(List\$,Delimiter\$,NewList\$)

Alphabetically sort a string list.

ReadRawBitmap(Image\$,Xsize,Ysize,Samples,Bits,Offset,Flags)

Read raw bitmap data into a bitmap in the PiXCL image list.

RawDataParamBox(RawImage\$,Help\$,Xsize,Ysize,Samples,Bits,Offset,Flags)

Display a dialog to enter the parameters for the ReadRawBitmap command.

Acos(Angle&,Value&)

Calculate the arc cosine of an angle in radians.

Asin(Angle&,Value&)

Calculate the arc sine of an angle in radians.

Atan(Angle&,Value&)

Calculate the arc tangent of an angle in radians.

Cos(Angle&,Value&)

Calculate the cosine of an angle in radians.

Cosh(Angle&,Value&)

Calculate the hyperbolic cosine of an angle in radians.

Sin(Angle&,Value&)

Calculate the sine of an angle in radians.

Sinh(Angle&,Value&)

Calculate the hyperbolic sine of an angle in radians.

Tan(Angle&,Value&)

Calculate the tangent of an angle in radians.

Tanh(Angle&,Value&)

Calculate the hyperbolic tangent of an angle in radians.

Average(Number&,.....,Value&)

Calculate the average of a list of numbers.

Exp(X&,Value&)

Calculate e^{**x}

Float(Number,FpNumber&)

Convert an integer into a floating point variable.

FpStr(FpNumber&,Number\$)

Convert a floating point number into a string.

FpVal(FpNumber\$,FpNumber&,Result)

Convert a string into a floating point variable.

Int(FpNumber&, Number)

Return the integer part of a floating point number.

Log10(FpNumber&,Value&)

Return the base 10 log of FpNumber&.

LogE(FpNumber&,Value&)

Return the Naperian log of FpNumber&.

Pow(Y&,X&,Value&)

Calculate the power function $Y^{**}X$

Sqrt(FpNumber&,Root&)

Calculate the square root of FpNumber&.

IDR_Closetdrisi(Result)

Close the IDRISI application window.

IDR_GetDataDir(IdrisiDataDir\$)

Get the current IDRISI data directory.

IDR_GetDir(IdrisiDir\$)

Get the IDRISI installation directory.

IDR_GetExtensions(Img\$,ImgDoc\$,Vec\$,VecDoc\$,Value\$,ValueDoc\$)

Get the current default IDRISI file extensions.

IDR_GetLanguage(Lang\$)

Get the current default language for IDRISI application and help.

IDR_IsPresent(Result)

Find out if there is an IDRISI application instance running.

IDR_Launch(TOKEN,Result)

Start IDRISI from a PiXCL application.

IDR_LaunchModule(ClientId,ClientOptions,ModName\$,CmdIn\$,OutputTitle\$,OutputUnits\$,PtHinst,ProcessID)

Start an IDRISI module and run it in the IDRISI client area.

IDR_RegisterClient(Client_ID)

Register an IDRISI client application. Values will be in the range 1-16.

IDR_SetDataDirectory(IdrisiDataDir\$,Result)

Set a new IDRISI data directory.

IDR_SetDebugMode(ON|OFF)

Enable or disable the IDRISI debug mode.

IDR_SetExtensions(Img\$,ImgDoc\$,Vec\$,VecDoc\$,Value\$,ValueDoc\$,Result)

Set the default IDRISI file extensions. Limited to three characters.

IDR_UnRegisterClient(Client_ID)

Unregister an IDRISI client application, and free the client number.

AddFont(Font\$,Result)

Add a new or custom font to the Windows font table.

RemoveFont(Font\$,Result)

Remove a font from the Windows font table.

GetTextSpacing(Spacing)

Get the current text spacing value. Used with the DrawText and DrawNumber commands.

SetTextSpacing(Spacing)

Set the current text spacing value. Used with the DrawText and DrawNumber commands.

Hypot(X&,Y&,Value&)

Calculate the hypotenuse of a right triangle fo sides X and Y.

SetMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the left mouse within a specified area.

SetCtrlMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the left mouse within a specified area, while holding down the Ctrl key.

SetShftMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the left mouse within a specified area, while holding down the Shift key.

SetDbIMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user double clicks the left mouse within a specified area.

SetRightMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the right mouse within a specified area. Right and Left mouse regions can overlap or overlay each other.

SetShftRightMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the right mouse within a specified area while holding down the Shift key. Right and Left mouse regions can overlap or overlay each other.

SetCtrlRightMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the right mouse within a specified area while holding down the Control key. Right and Left mouse regions can overlap or overlay each other.

SetDbfRightMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user double clicks the right mouse within a specified area.

SetMidMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the middle mouse within a specified area.

SetShftMidMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the middle mouse within a specified area while holding down the Shift key. Right, Left and Middle mouse regions can overlap or overlay each other.

SetCtrlMidMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user clicks the middle mouse within a specified area while holding down the Control key. Right, Left and Middle mouse regions can overlap or overlay each other.

SetDbIMidMouse(Region1_x1, Region1_y1, Region1_x2, Region1_y2, Label, x, y, . . .)

Sets up where the program branches to when the user double clicks the middle mouse within a specified area.

IDR_GetProgress(...)

(ClientID,ProcessID,Status,ReportType, ErrorFile\$, ErrorNumber,ErrorMessage\$,SubstString1\$,SubstString2\$,
Result_1, Result_2)

Gets the current progress details of an Idrisi client.

IDR_SetProgress(...)

(ClientID, ProcessID, Status, ReportType, ErrorFile\$, ErrorCode, ErrorMsg\$, Subst1\$, Subst2\$, Result_1, Result_2)

Sets the current progress details of an Idrisi client.

IDR_InitProgressTracking(ClientId,ProcName\$,ProcessID)

Starts the progress tracking of a registered Idrisi client.

FpAbs(Number&,Value&)

Calculate the absolute value of a floating point number.

WinHTMLHelp(File\$,MODE,TOKEN,Key\$,x1,y1,x2,y2)

Display HTML help (*.chm) in a window.

GetCPUInfo(CPUtype,NumCPU)

Returns the type and number of CPUs in the system.

ExportHistogram(ImageName\$,Result)

Read an image from the disk or from memory, and create channel histogram files, extension **.HST**.

DirListFiles(Path\$,Delimiter\$,Number,List\$)

Makes a delimited list of the contents of a directory.

ClipboardGetBitmap(ListImageName\$,Result)

Passes a PiXCL image list bitmap to the clipboard.

ClipboardPutBitmap(ListImageName\$,Result)

Passes an image in the clipboard to the PiXCL image list.

CopyWindowToClipboard(AppWin\$,ChildWin\$,Result)

Makes a copy of the window bitmap in the clipboard.

Switch-Case-Break-Default-EndSwitch

Switch (Integer_variable)

 Case <value>:

 ...

 Break

 ...

EndSwitch

PiXCL Command WinHelp() and Help Macro Commands.

The command `WinHelp(Helpfile$,COMMAND, Macro$)` can be used to pass any of the set of Help macro commands to the target help file. Not all macros will be useful with all help files.

Since it is necessary to include the quote character (") in many of the macro strings, you must first create some strings variables

```
Chr(34,Quote$)
QcomaQ$ = Quote$ + ", "
QcomaQ$ = QcomaQ$ + Quote$
```

These strings can then be used to create the necessary macros. For example

```
GetHelpMacro:
Macro$ = "CreateButton(" + Quote$
Macro$ = Macro$ + "IDM_NEW"
Macro$ = Macro$ + QcomaQ$
Macro$ = Macro$ + "Quit"
Macro$ = Macro$ + QcomaQ$
Macro$ = Macro$ + "Exit()"
Macro$ = Macro$ + Quote$
Macro$ = Macro$ + ")"

{this produces a help macro string
CreateButton("IDM_NEW","Quit","Exit()") }

HelpFile$ = "h:\p40tools\pdk\pixclhlp.hlp"
WinHelp(HelpFile$,COMMAND,Macro$)
Goto Wait_for_Input
```

will create an additional button on the target Help file, in this case a Quit button.

Some of the most commonly used Help Macros are:

| | |
|--|---|
| About() | Display a standard Help about box |
| CreateButton("button-id", "name", "button-macro") | Add a button to the Help file window menubar |
| DestroyButton("button-id") | Remove a button by ID. Useful when you defined the ID. See CreateButton. |
| DisableButton("button-id") | Disable a button by ID |
| EnableButton("button-id") | Enable a button by ID |
| ExecProgram("command-line", display-state) | Run a program from the help file. Display state 0 is visible, state 1 is minimized. |
| HelpOn() | Displays the Help file for the Windows Help application i.e. How to use Help... |
| HelpOnTop() | Toggle the Help window state between TOPMOST and NOTOPMOST. |
| History() | List the previous sequence of help file search commands. |
| InsertMenu("menu-id", "menu-name", | Insert a menu item into a help file |

menu-position)

JumpId("filename", "context-string")

Jump to a topic by context ID within the named help file.

Popupid("filename", "context-string")

Display the specified topic in a popup window

Print()

Print the current help topic. You could also add a "Print" button to the existing Help file to achieve the same result.

PrinterSetup()

Set up the current printer.

Search()

Display the search dialog box.

PiXCL Image Processing Extensions

PiXCL provides a comprehensive set of image processing commands for bitmaps displayed in the client area with the [DrawBitMap](#), [DrawSizedBitmap](#), [DrawTrBitmap](#), [DrawZoomedBitmap](#) and [DrawBitmapWindow](#) commands. We will refer to these generically as the [DrawBitmap](#) command set.

PiXCL creates a double linked list of all bitmaps loaded with the [DrawBitMap](#) command set, and this list references memory regions that contain the bitmap data. This means that if you are working with large or multiple bitmaps, Windows will use up copious amounts of memory. If there is inadequate space in memory, Windows will swap memory to the virtual memory region somewhere on your hard disk. The contents of the double linked list can be accessed with the [ListLoadedBitmaps](#) and [RenameListImage](#) commands.

PiXCL also in effect caches bitmap data, that is, if a specific filename has a bitmap table entry in PiXCL, and the file on the disk changes, these changes **WILL NOT** be reflected in the bitmap loaded into PiXCL, even if you issue another [DrawBitmap](#) command. What you have to do is first issue a [FreeBitmap\("name"\)](#) command, then reload the bitmap from disk with the [DrawBitmap](#) command. This is because PiXCL looks first at the list to see if a bitmap is already loaded, and if so, displays it direct from memory.

When you use these imaging commands, please note that they work on the current bitmap being referenced by the most recent [DrawBitmap](#) or image processing command. This bitmap will usually have been drawn somewhere into the visible client area space. Using one of the imaging commands will process the bitmap, but will not draw the bitmap into the client area. This requires an additional [DrawBitmap](#) command. The imaging processing commands have no effect on the PiXCL client area.

All image processing commands display a progress bar along the bottom of the client area as the function works. To disable this progress bar, use the [AutoProgressBar](#) command.

Descriptions for commands found in the more extensive **geoPiXCL** product only start with **geoPiXCL command**. All PiXCL commands are included in **geoPiXCL**.

Image Size Limitation when running under Windows 9x / ME:

The way that these operating systems assign global memory is flawed. If the request for a single block of memory exceeds 256MB, Windows 9x/ME denies the request, even if more than 256 MB of physical ram is installed in the PC. This means that the biggest image you can load (regardless of the amount of ram) is about 256 MB. If you need to work with larger images, we suggest that you move to NT4 or Windows 2000, which do not have the size limitation. In both cases you must have a suitable amount of virtual memory disk space assigned.

Related DrawBitmap Command Set:

[DrawBitmap](#) [DrawSizedBitmap](#) [DuplicateImage](#) [EnlargeImage](#) [EnlargeImageBox](#) [FreeBitmap](#) [FreeBitmapAll](#) [GetBitmapDim](#) [GetListBitmapDim](#) [ListLoadedBitmaps](#) [LoadBitmap](#)

Related image statistics and utility commands:

[CreatePALfile](#) [ConvertPALfile](#) [ExtractListImageRect](#) [InsertListImageRect](#) [LoadImageColormap](#) [SaveImageColormap](#) [Histogram](#) [UpdateHistogram](#) [ReportHistogramStats](#) [ShowHistogram](#) [ListLoadedBitmaps](#) [RenameListImage](#)

Blob Measurement commands:

In geoPiXCL 5.10 there is a new set of Blob commands which are used to identify and measure objects in an image.

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [ExportBlobObjectData](#) [FreeBlobEnv](#) [FreeBlobEnvAll](#) [FilterBlobObjects](#) [GetBlobCount](#) [ScanBlobObjects](#) [SetBlobFilterParams](#)

geoPiXCL Specific commands:

[BroveyEnhanceImage](#) [CombineThemes](#) [DecorrelStretchImage](#) [EditLANfileHeaderBox](#) [ExtractTrgAreaFiles](#) [GetThemeStats](#) [MakeNDVImage](#) [MakeScattergram](#) [MakeSpectralSignatureFiles](#) [MatrinTaylorMapping](#) [MLHClassify](#) [ModeFilterImage](#) [PCEnhanceImage](#) [PPDClassify](#) [ReadLANfileHeader](#) [ReadSPOTData](#) [ReadSPOTfileHeader](#) [RemapTheme](#) [WriteLANfileHeader](#)

geoPiXCL SHP/SHX "Shape" file commands:

[SHPAddAttribute](#) [SHPCreateSimpleObject](#) [SHPFileCreate](#) [SHPFileGetInfo](#) [SHPFileGetType](#) [SHPGetAttributeCount](#) [SHPGetFieldInfo](#) [SHPReadAttribute](#) [SHPReadObject](#) [SHPReadObjectVertices](#) [SHPWriteAttribute](#)

Alphabetical listing of the general image processing commands:

[AddNoiseToImage](#) [AverageImage](#) [AverageImageSet](#) [BlurImage](#) [CalibrateImage](#) [CombineChannels](#) [ComputeImage](#)

[ComputeImageBox](#) [ConvertColorSpace](#) [CropImage](#) [DespeckleImage](#) [EdgeDetectImage](#) [EmbossImage](#) [EqualizeImage](#)
[Filter5x5](#) [Filter15x15](#) [FlipImage](#) [GammaCorrectImage](#) [GaussianBlurImage](#) [GeoCorrectImage](#) [GeoTranslateImage](#)
[GetChannel](#) [InvertImage](#) [LinearEnhanceImage](#) [NLEnhanceImage](#) [NormalizeImage](#) [NormalizeImageRange](#) [OverlayImage](#)
[RemapImage](#) [ReplaceChannel](#) [ResampleImage](#) [ResizeImage](#) [RotateImage](#) [RotateImageExt](#) [ScaleCropImage](#)
[ScatterPixels](#) [SharpenImage](#) [SkewImage](#) [TuneImage](#)

AddNoiseToImage

[AddNoiseToImage](#) use a random number generator to add noise pixels to the current bitmap. Noise is often added to an image, followed by a smoothing filter process to sharpen detail.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32,

Syntax: [AddNoiseToImage\(Amount,type_TOKEN,Result\)](#)

Parameters:

[Amount](#) A number between 1 and 99. Larger and negative numbers are automatically converted to the absolute modulus 99 number.

[type_TOKEN](#) **NORMAL** adds noise generated by a normal function.
DISTRIBUTED adds noise generated by a distributed function.

[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

```
NoisyImage:  
    AddNoiseToImage(60, NORMAL, Res)  
    If Res <> 0 Then DrawBitmap(10,10, ImageFile$)  
    Goto Wait_for_Input
```

Related Commands

[DespeckleImage](#)

AverageImage

`AverageImage` uses a matrix to average a neighborhood of 25 pixels, resulting in a blurry, out of focus image.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32,

Syntax: `AverageImage(Amount,Result)`

Parameters:

`Amount` A number between 1 and 99. Larger and negative numbers are automatically converted to the absolute modulus 99 number.

`Result` Non-zero if the process was successful, otherwise 0.

Example:

Averaging:

```
AverageImage(60,Res)
If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
Goto Wait_for_Input
```

AverageImageSet

geoPIXCL command. A set of up to 16 images can be averaged to create a new image in the geoPIXCL image list.

Syntax: *AverageImageSet(Handle1,Handle2,Handle3,Handle4, ..., Handle16, AVRGIimage\$, Result)*

Parameters:

Handle1 .. Handle16 Up to 16 valid image handles. Set unused handles to 0.
AVRGIimage\$ The name of the average image created in the geoPIXCL image list.
Result 1 if the operation succeeded, otherwise 0.

Related Commands:

[AverageImage](#)

BlurImage

[BlurImage](#) uses a matrix to reduce the amount of contrast between pixels giving a smoothing or blurring effect.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [BlurImage\(Amount,Result\)](#)

Parameters:

[Amount](#) A number between 1 and 99. Larger and negative numbers are automatically converted to the absolute modulus 99 number.

[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

```
Blurring:
    BlurImage(80,Res)
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
    Goto Wait_for_Input
```

Related Command:

[GaussianBlurImage](#) , [AverageImage](#)

BroveyEnhanceImage

geoPIXCL command. The Brovey transform is one of the family of image enhancement algorithms designed to combine a high spatial information image (for example SPOT Panchromatic) with a multispectral lower spatial information image (for example Landsat TM).

The technique is commonly used to enhance Landat TM using SPOT Panchromatic, but is actually very good for other enhancements - sharpening SPOT XS with SPOT Pan, sharpening SPOT XS with orthoimages, combining geophysical data (for example Radiometrics) with SPOT Panchromatic, and so on. The algorithm is named after Bob Brovey, from Exxon.

The technique is as follows:

Assume two images, called TM and SPOT, the TM being a normal 7 band image, and the SPOT image being a SPOT Panchromatic image of the same area. The formula, for the common TM/SPOT Brovey Transform enhancement, is as follows:

```
RED   = (B5:1.65_um / (B2:0.56_um + B4:0.83_um + B5:1.65_um)) * B8:0.65_um_SPOT
GREEN = (B4:0.83_um / (B2:0.56_um + B4:0.83_um + B5:1.65_um)) * B8:0.65_um_SPOT
BLUE  = (B2:0.56_um / (B1:0.49_um + B4:0.83_um + B5:1.65_um)) * B8:0.65_um_SPOT
```

or better expressed ...

```
RED   = (Band5_TM / (Band2_TM + Band4_TM + Band5_TM)) * Band8_SPOT
GREEN = (Band4_TM / (Band2_TM + Band4_TM + Band5_TM)) * Band8_SPOT
BLUE  = (Band2_TM / (Band1_TM + Band4_TM + Band5_TM)) * Band8_SPOT
```

The BroveyEnhanceImage command in geoPiXCL provides the ability to specify a set of five images that are combined in the above manner.

Syntax: `BroveEnhanceImage(Handle1,Handle2, Handle4,Handle5, Handle8, NewImage$,Result)`

Parameters:

- Handle1 ... Handle8** The set of five loaded image handles. Handles are returned by a variety of commands such as [TuneImage](#), [SetCurrentBitmap](#) and [GetChannel](#).
- NewImage\$** The resulting 24-bit image that is created in the geoPiXCL image list. It can be displayed with [DrawBitmap](#), and saved to disk with the [SaveBitmap](#) commands.
- Result** 1 if the operation succeeds, otherwise 0. A common reason for failure is passing an invalid handle to the function.

Example:

BroveyEnhance:

```
LoadBitmap(Image1$,FULL)
LoadBitmap(Image2$,FULL)
LoadBitmap(Image4$,FULL)
LoadBitmap(Image5$,FULL)
LoadBitmap(Image8$,FULL)
SetCurrentBitmap(Image1$,FULL,Hndl_1)
SetCurrentBitmap(Image2$,FULL,Hndl_2)
SetCurrentBitmap(Image4$,FULL,Hndl_4)
SetCurrentBitmap(Image5$,FULL,Hndl_5)
SetCurrentBitmap(Image8$,FULL,Hndl_8)
UseCursor(WAIT)
DrawText(20,50,"Brovey Transform ... please wait!")
BroveEnhanceImage(Hndl_1,Hndl_2,Hndl_4,Hndl_5,Hndl_8,
    BroveyImage$,Res)
DrawBitmap(10,40,BroveImage$)
UseCursor(ARROW)
Goto Wait_for_Input
```

Related Commands:

[CalibrateImage](#) [ComputeImage](#) [SaveBitmap](#) [SetCurrentBitmap](#) [TuneImage](#)

Calibratelmage

A calibration function can be applied to images in the image list. The calibration is applied to the source image, or optionally, a new image can be created in the image list.

Syntax: `Calibratelmage(Handle,NewImage$,FnA&, FnB&, FnC&,Result)`

Parameters:

| | |
|-------------------------------------|---|
| <i>Handle</i> | The source image handle returned from a Tunelmage command. |
| <i>NewImage\$</i> | The name of a new image to create in the image list. Set this to a null string to overwrite the source image. |
| <i>FnA&, FnB&, FnC&</i> | Floating point arguments for $P_{out} = A * P_{in}^2 + B * P_{in} + C$ |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Remarks:

If *NewImage\$* is specified, the image is created in memory. This image will generally need to be saved with the SaveBitmap command. You may find the [RenameListImage](#) or [Duplicatelmage](#) commands useful as well.

Related Commands:

[Computelmage](#) [Tunelmage](#) [Duplicatelmage](#) [RenameListImage](#)

CombineChannels

This command provides a way to create color composites from a set of grayscale component images. Components must all have the same dimensions.

Syntax: `CombineChannels(RedHandle,GreenHandle,BlueHandle,
RGBHandle,Result)`

Parameters:

| | |
|--------------------|---|
| <i>RedHandle</i> | These are the handles of the component images, and are the stored as integer variables. You can get the variable by using one of the commands such as TuneImage , or SetCurrentBitmap , or GetChannel . |
| <i>GreenHandle</i> | |
| <i>BlueHandle</i> | |
| <i>RGBHandle</i> | The handle of the target color composite image. This must already exist, by displaying a color composite image, and using TuneImage to get the handle. See the sample code below. |
| <i>Result</i> | 0 if the command fails, otherwise it is the handle of the returned 24 bit color composite. |

Remarks:

The usual way that this command is used is to load the set of composite images into memory, using [DrawBitmap](#) or [DrawSizedBitmap](#). Remember that this loads the full bitmaps into memory, and a copy is written to the defined rectangle in the PiXCL client area.

Example:

In this code fragment example, the images are loaded into memory, handles acquired, then the new components are selected and combined, then displayed. The [TuneImage](#) commands have no effect, and provide a simple way to get the image handles. Note that output image handles will change when any geometric, color space or channel change command is issued. If in doubt, display the image and issue a [TuneImage](#) command.

```
Combiner: {draw all the test images}
    DrawBackGround
    DrawEdgeRectangle (sx1, sy1, sx2, sy2, SUNKENEDGE, ADJUST, RECT)
    DrawBitMap(23, 23, Image1$)
    TuneImage(0, 0, 0, 0, 0, 0, Band1Handle)
    DrawBitMap(23, 23, Image2$)
    TuneImage(0, 0, 0, 0, 0, 0, Band2Handle)
    DrawBitMap(23, 23, Image3$)
    TuneImage(0, 0, 0, 0, 0, 0, Band3Handle)
    DrawBitMap(23, 23, Image4$)
    TuneImage(0, 0, 0, 0, 0, 0, Band4Handle)
    DrawBitMap(23, 23, Image5$)
    TuneImage(0, 0, 0, 0, 0, 0, Band5Handle)
    DrawBitMap(23, 23, Image7$)
    TuneImage(0, 0, 0, 0, 0, 0, Band7Handle)
    DrawBitMap(23, 23, Image8$)
    TuneImage(0, 0, 0, 0, 0, 0, Band8Handle)

    CombineChannels (Band7Handle, Band3Handle, Band1Handle,
                    Band8Handle, Res)
    DrawBitMap(23, 23, Image8$)

    Goto Wait_for_Input
```

Related Commands:

[ReplaceChannel](#), [SaveBitmap](#)

CombineThemes

geoPiXCL command. Thematic images (8 bits / pixel) can contain various classes which sometimes need to be merged. With the CombineThemes command, you specify a set of input themes that are to be combined into a target theme. Within the theme image, the pixel values are changed, not the image colour map.

Syntax: *CombineThemes(ThemeImage\$, InputThemes[Index], TargetTheme, Result)*

Parameters:

| | |
|---------------------|---|
| <i>ThemeImage\$</i> | A theme image loaded into the geoPiXCL image list. |
| <i>InputThemes</i> | An integer array of appropriate size, as follows <i>[Index]</i> = number of input themes <i>[Index+1]</i> = theme #1 <i>[Index+n]</i> = theme #n |
| <i>TargetTheme</i> | The theme number (i.e. colour) to which the <i>InputThemes</i> will be converted. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[ModeFilterImage](#) [RemapTheme](#)

ComputeImage

Perform a computation on the pixels of a pair of images in the PiXCL image list. These images must be of the same lines and pixels dimensions, and have the same number of bits per pixel, or the operation will not occur.

Color Support: Grayscale, Indexed, RGB24, RGB32

Syntax: `ComputeImage(HandleA, HandleB, HandleC, opR,opG,opB, Divisor&,Bias&, clip_TOKEN, mode_TOKEN, bkg_TOKEN,NewImage$,Result)`

Parameters:

| | |
|-------------------------------|--|
| HandleA | Handle of the first image, also the destination for new pixel values if <i>NewImage\$</i> is a null string. <i>HandleA</i> is returned by a <code>TuneImage</code> call. |
| HandleB | Handle of second image of the pair. <i>HandleB</i> is returned by a <code>TuneImage</code> call. |
| HandleC | Mask image handle. This must be 8 bits per pixel only, or <i>Result</i> returns 0. If a mask is not being used, set this value to 0. <i>HandleC</i> is returned by a <code>TuneImage</code> call. |
| opR, opG, opB | R,G,B opacity values in the range 0-255, where 0 = 0% and 255 = 100%. Opacity defines the amount of the <i>HandleB</i> image that is merged with the complementary amount of the <i>HandleA</i> image. Hence, for opacity =255 (i.e. 100%), all of image B is used. If opacity = 64, then 25% of image B is combined with 75% of image A. <i>opR, opG, opB</i> do not have to have the same values. |
| Divisor&,Bias& | Floating point scaling values: where $P1 = P0/Divisor\& + Bias\&$. Default values are 1.0, 0.0. These numbers can be negative. The scaling operation is performed after the compute image function (see <code>mode_TOKEN</code>), and before any <code>CLIP/NOCLIP</code> operation. <i>Divisor&</i> and <i>Bias&</i> have no meaning fo composite and blend operations, and the values are ignored. |

CLIP | NOCLIP Token that indicates whether resulting pixel values are clipped between 0 and 255. If you select the NOCLIP option, 8 bit colour values wrap around. While this can lead to interesting looking images, you must decide if the operation is valid for your project and data.

| | |
|-------------------|--|
| mode_TOKEN | Defines the type of image arithmetic operation, as follows: |
| ADD | Add the pixel values of <i>HandleA</i> and <i>HandleB</i> images. |
| SUBTRACT | Subtract pixel values of <i>HandleB</i> from <i>HandleA</i> images. |
| DIFFERENCE | Difference of the pixel values between <i>HandleA</i> and <i>HandleB</i> |
| COMPOSITE | Composite of <i>HandleA</i> and supplied mask <i>HandleC</i> . |
| BLEND | Blend the pixels values of <i>HandleA</i> and <i>HandleB</i> using mask <i>HandleC</i> . |
| MULTIPLY | Multiply pixels and divide the result by 256. |
| LIGHTER | Destination pixel is the lighter of <i>HandleA</i> and <i>HandleB</i> . |
| DARKER | Destination pixel is the darker of <i>HandleA</i> and <i>HandleB</i> . |

Additional mode_TOKENs for geographic data sets.

| | |
|--------------|---|
| RATIO | Destination pixel is the ratio of <i>HandleA</i> / <i>HandleB</i> . |
| MINRATIO | Destination pixel is the minimum ratio of $(HandleA - \min(HandleA)) / (HandleB - \min(HandleB))$ |
| DIFFOVERSUM | Difference Over Sum. $(HandleA - HandleB) / (HandleA + HandleB)$ |
| ROOTSUMOFSQR | Root Sum Of Squares. |
| BIOMASS | Biomass calculation. |
| NDVI | Normalized Difference Vegetation Index calculation. |

| | |
|------------------|---|
| bkg_TOKEN | Set the background color for all operations. This is primarily used for operations where the opacity values are less than 100%. |
| BKGNDBLACK | Sets the target image background to black (0,0,0) before the image computation. |
| BKGNDGRAY | Sets the target image background to gray (128,128,128) before the image computation. |
| BKGNDWHITE | Sets the target image background to white (255,255,255) before the image computation. |
| BKGNDOVERLAY | Sets the target image background to the <i>HandleA</i> image before the image computation. |
| BKGNDMASK | Use the mask image <i>HandleC</i> . |
| BKGNDMASKINV | Invert the mask image operation. |

NewImage\$ Name of the new image to be created. Set this to a `path\name` value, or specify an image already loaded in the PiXCL image list. If *NewImage\$* is null, the image specified by *HandleA* is the destination image. *NewImage\$* can be saved to disk with the `SaveBitmap` command. You can check the contents of the PiXCL image list with the `ListLoadedBitmaps` command.

Result 1 if the function completed successfully, otherwise 0.

Related Commands:

[ComputeImageBox](#) [ListLoadedBitmaps](#) [SaveBitmap](#) [TuneImage](#)

ComputeImageBox

Displays a dialog box for the manual entry of image arithmetic operations. All parameters MUST be string, integer or floating point variables, or the command will fail with a syntax error. The variables returned are usually passed to a [ComputeImage](#) command.

Syntax: `ComputeImageBox(Image1$,Image2$,Mask3$,Help$,Function,RedOpacity, GreenOpacity, BlueOpacity,Divisor&,Bias&,ClipMode,Result)`

Parameters:

| | |
|---|---|
| Image1\$,Image2\$ | Input images. The combobox dropdown list displays all the images loaded in the PiXCL image list. This means that you must first load the images with LoadBitmap or DrawBitmap . |
| Mask3\$ | Mask image, if required. If not selected, returns a null string. |
| Help\$ | Help string displayed in a MessageBox when the Help button is pressed. |
| Function | Returned value that can be passed to the ComputeImage function. |
| RedOpacity | 0-255 value for red opacity. 0 = 0%, and 255 = 100%. |
| GreenOpacity | 0-255 value for red opacity. 0 = 0%, and 255 = 100%. |
| BlueOpacity | 0-255 value for red opacity. 0 = 0%, and 255 = 100%. |
| Divisor&, Bias& | Defaults are 1.0 and 0.0. |
| ClipMode | 0 = No clip, 1 = clip result pixel values between 0 and 255. |
| Result | 1 if the operation succeeded, otherwise 0. |

Related Commands:

[ComputeImage](#)

ConvertColorSpace

Bitmaps loaded and displayed by PiXCL commands are in one of several possible colorspaces, a term that is related to the number of bits per pixel per composite color. You are most likely to be familiar with the standard additive **Red-Green-Blue** method used by your color monitor, and the subtractive **Cyan-Magenta-Yellow** method used in color negative film and color printing.

PiXCL provides the means to convert the colorspace of the current bitmap image into other formats for more suitable processing. For example, you can load and display an 8 bit color image, turn it into a 16, 24 or 32 bit image in memory, replace or modify a channel, then convert back to 8 bit colorspace. All images in memory can be saved to disk using the [SaveBitmap](#) or [SaveBitmapHandle](#) command.

Syntax: [ConvertColorSpace](#)(COLORSPACE_token,DITHER_token,Result)

Parameters:

[COLORSPACE_token](#) Defines the target colorspace. The available options are

| | |
|---------------------------|--|
| RGB32 | 32 bit with the high 8 bits ignored. |
| CMYK | 32 bit C yan, M agenta, Y ellow and blacK separations. |
| RGB24 | Standard 24 bit color composite. |
| CIELAB | CieLab 24 bit color mode. |
| RGB555 | A 16 bit composite. The 16th bit is ignored. |
| RGB565 | A 16 bit composite. |
| GRAY16 | 16-bit linear grayscale black=0, white=65536. |
| INDEXED | Standard 8-bit colormapped image mode. |
| GRAYSCALE | Standard linear grayscale black=0, white=255. |
| MONO | 1 bit per pixel monochrome. |

[DITHER_token](#) Defines the target colorspace dither method, if applicable.

| | |
|-------------------------|-------------------------------------|
| NONE | no dithering. |
| ORDERED | dither using Floyd-Steinberg method |
| DIFFUSE | dither using error diffusion. |

[Result](#) 0 if the operation failed, otherwise is the new handle of the image. You will need to redraw the image to see the result.

Remarks:

You cannot convert between [RGB555](#) and [RGB565](#) without converting to [RGB24](#) first. If the existing color mode is the same as the new mode, no operation is performed, but [Result](#) returns a valid handle.

To convert to CMYK separations, you must first convert to RGB32. If you don't first convert to RGB32, [Result](#) returns 0.

The Cyan, Magenta and Yellow channels in a CMYK are not the same as inverted RGB. The black channel is the minimum gray level that is common to all channels. This is the efficient method that the printing industry uses to minimize the use of expensive colour inks, and maximize the use of inexpensive black inks.

The CMYK algorithm first inverts the colours of each pixel to get standard CMY . . .

```
pRed = 255 - pRed;  
pGreen = 255 - pGreen;  
pBlue = 255 - pBlue;
```

then calculates and removes the gray component into the black channel . . .

```
pK = min(pRed,pGreen,pBlue);  
pC = pRed - pK;  
pM = pGreen - pK;  
pY = pBlue - pK;
```

Using [DrawBitmap](#) on a CMYK separation will result in a dark image, because firstly the colours have been inverted and then the common gray value subtracted.

Related Commands:

[SaveBitmap](#) [ListLoadedBitmaps](#) [DrawBitmap](#)

CreateBlobEnv

geoPIXCL command. A blob environment is required to identify, count and measure Binary Large Objects (Blobs) in an image. These blobs will typically have been identified by using PPDClassify or MLHClassify, both of which produce a 256 colour theme file in the geoPIXCL image list.

Syntax: `CreateBlobEnv(ThemeFile$,SQMM|SQINCH|PIXEL,PixelArea&,BlobEnvID)`

Parameters:

| | |
|--------------------------------|--|
| ThemeFile\$ | The name of a 256 colour image loaded into the list. |
| SQMM | Blob measurements are in square millimeters per pixel. |
| SQINCH | Blob measurements are in square inches per pixel. |
| PIXEL | Blob measurements are in pixels. |
| PixelArea& | The area of a single pixel according to the above token. This value is related to the resolution of the scanner, if this is the source of the image to be measured. For example, at 72dpi (i.e. 72 pixels per inch), one pixel is 0.0138 inches or 0.353mm per side, hence the pixel area (assumed square pixels) is 0.0002 sq.inches or 0.1245 sq.mm For PIXEL metric, set this to 1.0. This value is used to report blob size and length. |
| BlobEnvID | The unique 0 indexed number that identifies the new Blob environment for ThemeFile\$. An image can have one blob environment at a time, but any image loaded into the list can have a unique BlobEnvID . BlobEnvID gets reset to 0 if a FreeBlobEnvAll command is issued. |

Remarks:

A typical sequence of operations that makes use of the blob command set might be

1. Load an image with [LoadBitmap](#) or [DrawBitmap](#)
2. Decide on class boundaries and create a theme file with [PPDClassify](#)
3. Overlay the them on to the original image with [OverlayImage](#)
4. Create a blob environment with [CreateBlobEnv](#)
5. Identify the blobs with [ScanBlobObjects](#)
6. Set the filter parameters with [SetBlobFilterParams](#) and apply with [FilterBlobObjects](#).
- 7.

Related Commands:

[FreeBlobEnv](#) [FreeBlobEnvAll](#) [DrawBlobObjLabels](#) [ExportBlobObjectData](#) [FilterBlobObjects](#) [GetBlobCount](#)
[GetBlobObjectData](#) [ScanBlobObjects](#) [SetBlobFilterParams](#)

CreateScattergram

geoPiXCL command. You can create a scattergram bitmap from two 8-bit images of the same dimensions, if the images are loaded into the geoPiXCL image list. The result is a 256x256x8-bit image stored in the PiXCL image list, called "SourceDir\$\temp.bmp". You can optionally specify a mask image to produce a scattergram only certain regions of the X and Y-axis images.

Syntax: *CreateScattergram(X-axisImage\$, Y-axisImage\$, MaskImage\$, MaskString\$, Result)*

Parameters:

| | |
|---------------------|---|
| <i>XaxisImage\$</i> | The 8-bit image that is to be used for the X-axis. |
| <i>YaxisImage\$</i> | The 8-bit image that is to be used for the Y-axis. |
| <i>MaskImage\$</i> | The 8-bit image that is to be used for mask. Set to "" (NULL) if no mask is required. |
| <i>MaskString\$</i> | The string that defines the entries used for the mask. Ignored if <i>MaskImage\$</i> is NULL. |
| <i>Result</i> | 1 if the scattergram bitmap is created, otherwise 0. |

Remarks:

The scattergram bitmap can be saved to disk with the SaveBitmap command.

MaskString\$ is in the form "1,2,4-7,9,11-17", with no spaces. These are the index values of the mask image which is expected to be the result of a classification or a manually generated mask image.

Example:

See the scatgrm.pxl sample code.

Related Commands:

CropImage

[CropImage](#) extracts a rectangular section of the current bitmap, discards the original bitmap and sets the smaller section as the current bitmap.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [CropImage\(x1,y1,x2,y2,Result\)](#)

Parameters:

[x1,y1,x2,y2](#) The coordinates in the image that define the corners of the desired region.

Result Non-zero if the process was successful, otherwise 0.

Example:

[Cropper:](#)

```
CropImage(20,20,180,220,Res)
If Res <> 0
    DrawBackGround
    DrawBitMap(23,23,ImageFile$)
Endif
Goto Wait_for_Input
```

Related Commands:

[ResampleImage](#) [ScaleCropImage](#)

DecorrelStretchImage

geoPiXCL command. The decorrelation stretch enhancement is related to the Principal Component enhancement, and can be considered to be a sort of inverse. The visual result is that the original RGB colour space utilisation is maximised. Note the SIG files input must be the same set that was used to create the PCA files originally.

Syntax: *DecorrelStretchImage(SigFile\$, BaseFile\$, Components, MODE_token, Result)*

Parameters:

| | |
|-------------------|---|
| <i>SigList\$</i> | A list of the SIG files that were previously created from a corresponding set of TRG files. |
| <i>BaseFile\$</i> | The name that is used to create the output channel names. The last four characters of the name are replaced with PCAn, where 'n' is the component number starting from 1. For example, if the input is "brsfbd1.bmp", the first PC file will be "brsfpca1.bmp". Note that only BMP and TIF/geoTIF formats are supported at present. |
| <i>Components</i> | The number of principal components channels to be created. This is <= to the number of input bands specified in the SIG file. |
| <i>MODE_Token</i> | Sets the processing mode. LISTADD creates the output files in the image list. DISKONLY creates the output files on disk. |
| <i>Result</i> | 1 if the operation was succesful, otherwise 0. |

Related Commands:

[PCEnhanceImage](#)

DespeckleImage

[DespeckleImage](#) uses a matrix to try to remove high frequency noise pixels in the image. For example, an image may include substantially brighter and darker single pixels, visually often referred to as a “salt-and-pepper” effect. In color images this may appear as seemingly random color pixels. You can simulate this effect by adding random noise to an image with the [AddNoiseToImage](#) command, then use the [DespeckleImage](#) command to clean up most of the added effect.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [DespeckleImage](#)(*Amount*, *Result*)

Parameters:

[Amount](#) The amount of despeckle effect to use, range 0 - 50.
[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

```
EdgeDetect:  
    DespeckleImage(17, Res)  
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)  
    Goto Wait_for_Input
```

Related Commands:

[AddNoiseToImage](#)

DrawBlobObjects

geoPIXCL command. Once blob objects have been identified in an image, each blob has a unique ID number starting at 1, and can be filtered for size and length. These blobs can be rendered into a blob overlay image (8 bits per pixel) in the colour of choice, according to the image colour map. If the overlay is already displayed, for example in a bitmap window, it needs to be redrawn.

Syntax: [DrawBlobObjects\(BlobEnvID, DRAW_Token, ThemeColour, ThemeImage\\$, Result\)](#)

Parameters:

| | |
|----------------------------------|--|
| BlobEnvID | A number that identifies the blob, returned from CreateBlobEnv . |
| ALL_OBJ | Render all blob objects. |
| FILTERED_OBJ | Render a subset of blob objects |
| ALL_OBJ_RTN | Render all blob objects, retaining the original image data. |
| FILTERED_OBJ_RTN | Render a subset of blob objects, retaining the original image data. |
| ThemeColour | The theme colour index in the range 0-255 that is used to render the blobs. The actual colour is defined in the ThemeImage\$ colour map. |
| ThemeImage\$ | The theme image loaded into the geoPIXCL image list. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[DrawBlobObjLabelsFreeBlobEnv](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#) [GetBlobCount](#)
[GetBlobObjectData](#) [ScanBlobObjects](#) [SetBlobFilterParams](#)

DrawBlobObjLabels

geoPIXCL command. Once blob objects have been identified in an image, each blob has a unique ID number starting at 1. These numbers can be rendered into a blob overlay image (8 bits per pixel) in the colour of choice, according to the image colour map. If the overlay is already displayed, for example in a bitmap window, it needs to be redrawn. See the [OverlayImage](#) command to do a transparent overlays on to a target image.

Syntax: `DrawBlobObjLabels(BlobEnvID, ALL_OBJ|FILTERED_OBJ, LabelColour, ThemeImage$, Result)`

Parameters:

| | |
|------------------------------|---|
| BlobEnvID | A number that identifies the blob environment, returned from CreateBlobEnv . |
| ALL_OBJ | Render all blob object ID labels. |
| FILTERED_OBJ | Render a subset of blob object ID labels according to filter parameters. |
| LabelColour | The label colour index in the range 0-255. The actual colour is defined in the ThemeImage\$ colour map. |
| ThemeImage\$ | The theme image loaded into the geoPIXCL image list. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[DrawBlobObjects](#) [FreeBlobEnv](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#) [GetBlobCount](#)
[GetBlobObjectData](#) [ScanBlobObjects](#) [SetBlobFilterParams](#)

EditLANfileHeaderBox

geoPiXCL command. This command displays a dialog that reads an Erdas™ LAN/GIS file header, and enables the header details to be edited and saved.

Syntax: [EditLANfileHeaderBox\(x1,y1,LANImage\\$,Result\)](#)

Parameters:

| | |
|----------------------------|--|
| x1,y1 | The top left corner client area coordinate used to position the dialog box. |
| LANImage\$ | The LAN/GIS file to read. |
| Result | 1 if the header information is saved (Save button) or 0 if the Cancel button is pressed. |

Related Commands:

[ReadLANfileHeader](#)

EdgeDetectImage

[EdgeDetectImage](#) uses a matrix to find high areas of contrast in an image and enhance them, leaving only the 'edges'.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [EdgeDetectImage\(Result\)](#)

Parameters:

[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

```
EdgeDetect:  
    EdgeDetectImage(Res)  
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)  
    Goto Wait_for_Input
```

EmbossImage

`EmbossImage` uses a matrix to give the effect of an embossing.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `EmbossImage(Amount,Result)`

Parameters:

Amount A number between 1 and 7. Larger and negative numbers are automatically converted to the absolute modulus 7 number.

Result Non-zero if the process was successful, otherwise 0.

Example:

Embossing:

```
EmbossImage(4,Res)
If Res <> 0 Then DrawBitmap(10,10,ImageFile$)
Goto Wait_for_Input
```


EqualizeImage

`EqualizeImage` creates a histogram from the pixel values in the current bitmap and equalizes the range of values in the histogram. The histogram is then remapped to the current bitmap. The histogram creation is not reported in the progress reporting since it does not take long to create.

Syntax: `EqualizeImage(Result)`

Parameters:

Result Non-zero if the process was successful, otherwise 0.

`Equalizer:`

```
EqualizeImage(Res)
If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
Goto Wait_for_Input
```

ExportBlobObjectData

geoPIXCL command. A blob environment contains a set of BlobObjects, whose details can be exported to a text file. Size and length are reported in the current metric used in [CreateBlobEnv](#).

Syntax: `ExportBlobObjectData(BlobEnvID, ALL_OBJ|FILTERED_OBJ,ExportFile$,Result)`

Parameters:

| | |
|------------------------------|--|
| BlobEnvID | A number that identifies the blob, returned from CreateBlobEnv . |
| ALL_OBJ | Report on all blob objects. |
| FILTERED_OBJ | Report on a filtered subset of the blob objects only. |
| ExportFile\$ | The name of the export file. If the file already exists, it is overwritten. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#) [GetBlobCount](#)
[ScanBlobObjects](#) [SetBlobFilterParams](#)

ExtractTrgAreaFiles

geoPIXCL Command. An 8-bit image containing training areas has been loaded into memory, and its handle is passed into the function, along with the training area base filename, and the number of desired training areas.

Each pixel in the image is checked to see if it is non-zero, which means it is part of training area. The number of training areas defines the range of pixel values to be checked. These MUST be contiguous in the range 1 to #training_areas.

The ASCII TRG file line format is defined as

```
line# pixel# run_length#\r\n
```

Syntax: [ExtractTrgAreaFiles\(Handle,TRGBaseName\\$,Classes,MLH|PCA,SkipRes\)](#)

Parameters:

| | |
|-------------------------------|--|
| Handle | The handle of the bitmap that contains the training area. |
| TRGBaseName\$ | The name that is used to create area files. This is generally the same one of the source images to be used later for classification or enhancement. |
| Classes | The number of class training areas to be extracted from the source bitmap. |
| MLH PCA | Maximum likelihood or Principal component mode. |
| SkipRes | Numeric variable that has no input relevance to the MLH mode, and defines the line skip factor for creating PCA 'n'th line training area files. On return, SkipRes holds either the number of ascii TRG files created, or 0 if the operation fails. |

Remarks:

You can define training areas for the a [PCA](#) mode, to set specific regions of interest for later enhancement. If the mask file specified is not 8 bit, [SkipRes](#) returns 0.

Related Commands:

[MakeSpectralSignatureFiles](#)

FilterBlobObjects

geoPIXCL command. A blob environment containing blob objects can be filtered for size, length and edge-rejection. Edge-rejection identifies blobs with pixels on the edges of the image that may be larger, and so should not be counted. Use the [DrawBlobObjects](#) and [DrawBlobObjLabels](#) commands to render a new overlay.

Syntax: `FilterBlobObjects(BlobEnvID,LENGTH|SIZE| EDGEREJECT,Result)`

Parameters:

| | |
|----------------------------|--|
| BlobEnvID | A number that identifies the blob, returned from CreateBlobEnv . |
| LENGTH | Filter Countable blobs according the min and max length parameters, retaining only those within the range, and setting all others to NonCountable. |
| SIZE | Filter according the min and max size parameters, retaining only those within the range, and setting all others to NonCountable. |
| EDGEREJECT | Set all blobs with pixels on the edge of the image to NonCountable. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnv](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [GetBlobCount](#)
[ScanBlobObjects](#) [SetBlobFilterParams](#)

Filter5x5, Filter15x15

PiXCL 4.12 and later provides user defined filter operations with either 5x5 or 15x15 matrices, which are imported from text files in a standard format described below. A matrix can be used to sharpen, emboss, blur and detect the edges of an image.

Filter files have the following format: two lines describing the BIAS and DIVISOR, follow by either a 5x5 or 15x15 matrix of positive or negative integer values, delimited within each line by at least ONE space character (0x20) or TAB character (0x09), and with a carriage-return / line-feed delimiting the end of each line, with the exception of the last line, which MUST terminate with an End-Of-File character. This means that filter files can be created and edited with Notepad or the PiXCL MDI editor. The default file extension for PiXCL filter files is **.mtx**. For example ...

```
BIAS 30
DIVISOR 15
2 0 -1 0 2
0 1 0 1 0
0 0 3 1 0
0 1 0 1 0
2 0 0 0 2
```

this filter could be saved as **test5x.mtx**.

The general format of a 15x15 filter is the same as the 5x5. "+" signs are not required. For edge pixels, the neighbourhood outside the bitmap coordinates are considered to be zero. Hence for a 5x5 filter, a 2 pixel wide border will have slightly different results than pixels within the body of the bitmap. For a 15x15, this border will be seven pixels wide.

The steps taken to perform a matrix convolution are:

1. Neighbourhood pixel values and corresponding matrix values are multiplied together.
2. The resulting values are added together for each pixel.
3. The sum is then divided by the factor (which must not be zero).
4. The bias is added to the final pixel values.

Prior to step #1, pixel values are multiplied by 16384, and after step #4, the result is divided by 16384. This integer method is just as accurate and is faster than doing the calculations in floating point.

The related [Create5x5Filter](#) command displays a dialog in which a 5x5 filter can be created or edited. 15x15 filters can be created with a text editor, such as the PiXCL MDI editor or Notepad.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `Filter5x(FilterName$, Result)`
`Filter15x(FilterName$, Result)`

Parameters:

FilterName\$ The name of the import filter.

Result The filtered image handle, otherwise zero. A zero result can occur if the filter file is invalid in some way, such as a trailing carriage-return character

Remarks:

When any filter is processing an image, a progress bar is displayed along the bottom of the PiXCL application window, which will overwrite a status bar if one is present. Please be aware that even on small images, a 15x15 matrix will be quite slow, as the operation is computationally complex, unless there are many matrix elements that are zero.

This will often occur when you want to pass a smaller matrix kernel smaller than 15x15 but larger than 5x5. The desired kernel of values are set, with the surrounding values all set to zero.

Examples:

The following code fragments cause the current image to be processed with an imported custom filter matrix.

```
Do5xFilter:
    WaitInput(1) {allow Toolbar button to process}
    If ImageFile$ <> ""
```

```
Filter5x$ = SourceDir$ + "\test5x.mtx"
UseCursor(WAIT)
FilterImage5x(Filter5x$,Res)
UseCursor(ARROW)
If Res <> 0 Then DrawBitmap(30,38,ImageFile$)
Endif
Goto Wait_for_Input

Do15xFilter:
WaitInput(1)
If ImageFile$ <> ""
Filter15x$ = SourceDir$ + "\average.mtx"
UseCursor(WAIT)
FilterImage15x(Filter15x$,Res)
UseCursor(ARROW)
If Res <> 0 Then DrawBitmap(30,38,ImageFile$)
Endif
Goto Wait_for_Input
```

Related Commands:

[Create5x5Filter](#)

FlipImage

Flip a target bitmap vertically or mirror horizontally .

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `FlipImage(VERT|HORZ,Result)`

Parameters:

`VERT|HORZ` Vertical or horizontal token.

Result Non-zero if the process was successful, otherwise 0.

Example:

```
FlipVertical:
    FlipImage(VERT,Res)
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
    Goto Wait_for_Input
FlipHorizontal:
    FlipImage(HORZ,Res)
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
    Goto Wait_for_Input
```

FreeBlobEnv

geoPIXCL command. A blob environment can be removed from memory once it is no longer required.

Syntax: [FreeBlobEnv\(BlobEnvID\)](#)

Parameters:

[BlobEnvID](#) A number that identifies the blob, returned from [CreateBlobEnv](#).

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#) [GetBlobCount](#)
[GetBlobObjectData](#) [ScanBlobObjects](#) [SetBlobFilterParams](#)

FreeBlobEnvAll

geoPIXCL command. All blob environments can be removed from memory once they are no longer required. **geoPIXCL** removes all blob environments automatically when it terminates.

Syntax: [FreeBlobEnvAll](#)

Parameters:

None.

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnv](#) [ExportBlobObjectData](#) [FilterBlobObjects](#) [GetBlobCount](#)
[GetBlobObjectData](#) [ScanBlobObjects](#) [SetBlobFilterParams](#)

FreeChannel

If you use the [GetChannel](#) command, you MUST eventually use the [FreeChannel](#) command.

Syntax: [FreeChannel\(Handle,Result\)](#)

Parameters:

[Handle](#) A non-zero number returned from the [GetChannel](#) command.

[Result](#) 1 if the channel image memory has been freed, otherwise 0.

Related Commands:

[GetChannel](#)

GammaCorrectImage

All image formats have at least one thing in common: a higher pixel number means a brighter color value. This rule has two corollaries. Firstly, the highest combination of values possible for a pixel produces pure white; and secondly the lowest combination of values possible for a pixel produces pure black.

It would follow logically that a value half-way up any scale would have a luminance exactly between black and white. This assumes that our hypothetical scale would allow an integer value exactly in the middle, which in reality it does not, since image formats are based on powers of two, and integer midpoints are not possible.

Consider an approximate midpoint in a 256 color grey scale format.. This approximate midpoint - say 120 - should produce an identical luminance on the monitor when the original image and the viewed image are compared.

A computer monitor displays colors by exciting the screen phosphors which unfortunately do not excite linearly. For example, if a computer reads a luminance value from a photographic image and sends it directly to the monitor, the displayed color will be dimmer than in the original photograph.

This is where gamma correction is used: a gamma correction value adjusts for the non-linearity of phosphor excitation. By selecting an appropriate gamma value, our approximate midpoint of 120 produces a monitor value of 168.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `GammaCorrectImage(GammaR, GammaG, GammaB, Result)`

Parameters:

GammaR|G|B A positive number between 100 and 999, which is the gamma value x100. E.g. a gamma of 2.45 is written as 245.

Result Non-zero if the process was successful, otherwise 0.

Example:

```
GammaCorrection:
  UseCursor(WAIT)
  GammaCorrectImage(220,210,233,Res)
  If Res <> 0 Then DrawBitMap(23,23,ImageFile$)
  UseCursor(ARROW)
  Goto Wait_for_Input
```

GaussianBlurImage

[GaussianBlurImage](#) applies a Gaussian function on an image that can blur an image greater than [AverageImage](#) or [BlurImage](#). The effect is also quite visually different. The blurring in the [GaussianBlurImage](#) function is also exponential and 99 is considered the most usable maximum, although the limit is much higher. A blur value of 0 has no effect on the image.

Processing time for Gaussian blur can be quite long, so the cursor changes to an hourglass, and a progress bar appears along the bottom of the PiXCL application window.

Gaussian blur is used as the first step to sharpen an image. For example, a blur factor of about 10 will smooth out pixel values, especially adjacent pixels that are marked different in value such as noise. Scanned photos often exhibit this effect. Next, apply a sharpen function. This will often result in a visually more appealing image than the original with just the sharpen function applied.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [GaussianBlurImage\(Amount,Result\)](#)

Parameters:

[Amount](#) A number between 1 and 99. Larger and negative numbers are automatically converted to the absolute modulus 99 number.

[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

[GBlurring:](#)

```
UseCursor(WAIT)
GaussianBlurImage(90,Res)
If Res <> 0 Then DrawBitMap(23,23,ImageFile$)
UseCursor(ARROW)
Goto Wait_for_Input
```

GeoCorrectImage

PiXCL 4.2 and later provide a simple geometric correction capability, which is also sometimes referred to as image warping. The geometric operation uses an affine transform to distort the image according to four control points, generally chosen or computed to the corner points of the image. This correction provides a means to register one image onto another under program or user control.

The size of the image before and after the transformation is unchanged, which means that the resulting image may have black areas that do not have valid pixel data as a result of the geometric transformation. Portions of the original image data may not be retained.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `GeoCorrectImage(x1,y1,x2,y2,x3,y3,x4,y4,Mode_TOKEN,Result)`

Parameters:

`x1,y1,x2,y2,x3,y3,x4,y4` The four new corner points that define the image transformation.

`Mode_TOKEN` **NEAREST:** nearest neighbour resampling

`BILINEAR:` bilinear interpolation resampling

`BICUBIC:` bicubic interpolation resampling

`Result` 1 if the operation was successful, otherwise 0.

Example:

See the `geocorrect.pxl` sample application.

Related Commands:

[GeoTranslateImage](#) [ResampleImage](#) [RotateImage](#) [RotateImageExt](#)

GeoTranslatelImage

PiXCL 4.2 and later provide a geometric translation capability, using an affine transform to translate the image according to the control point that defines a new corner point. This correction provides a means to move the origin of one image to another under program or user control.

The size of the image before and after the transformation is unchanged, and translated regions outside of the image dimensions are not retained. The resulting bitmap is still in memory, and can be saved with the [SaveBitmap](#) command.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [GeoTranslatelImage\(zx1,zy1, Mode_TOKEN,Result\)](#)

Parameters:

[zx1,zy1](#) The new corner point that defines the image transformation. The image origin (0,0) is assumed to be the top-left corner. Hence, to translate the top-left corner pixel 100 pixels in the X and Y axes, [Zx1, Zy1](#) = 100,100.

[Mode_TOKEN](#) **NEAREST:** nearest neighbour resampling.
BILINEAR: bilinear interpolation resampling
BICUBIC: bicubic interpolation resampling

[Result](#) 1 if the operation was successful, otherwise 0.

Example:

See the [geocorrect.pxl](#) sample application.

Related Commands:

[GeoCorrectImage](#) [ResampleImage](#) [RotatelImage](#) [RotatelImageExt](#)

GetBlobCount

geoPIXCL command. A blob environment contains blob objects that are by default set to Countable state. In the process of Filtering blobs for Length or Size, some may be set to Non-Countable state.

Syntax: *GetBlobCount(BlobEnvID, TotalBlobs, FilteredBlobs)*

Parameters:

BlobEnvID A number that identifies the blob, returned from [CreateBlobEnv](#).
TotalBlobs The total number of blob objects in the environment.
FilteredBlobs The number of Countable blob objects in the environment.

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnv](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#)
[GetBlobObjectData](#) [ScanBlobObjects](#) [SetBlobFilterParams](#)

GetBlobObjectData

geoPIXCL command. A blob environment contains blob objects that contain measurement data. Use the [GetBlobObjectData](#) command to retrieve blob information.

Syntax: [GetBlobObjectData](#)(*BlobEnvID*, *BlobObjectID*, LENGTH|SIZE,*Measurement*)

Parameters:

| | |
|------------------------------|---|
| BlobEnvID | A number that identifies the blob, returned from CreateBlobEnv . |
| BlobObjectID | The blob object identifier that contains the desired measurement. If the BlobObjectID is negative or greater than the number of blob objects, Measurement returns -1. |
| LENGTH | Returns the length of the blob in pixels. |
| SIZE | Returns the size of the blob in pixels. |
| Measurement | The returned measurement value. |

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnv](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#) =
[ScanBlobObjects](#) [SetBlobFilterParams](#)

GetChannel

This command provides the means to get the handle of the current bitmap, or a color channel of the current bitmap.

Syntax: `GetChannel(RED|GREEN|BLUE, Handle)`

Parameters:

`RED|GREEN|BLUE` The color channel to be accessed

Handle The returned handle of the current bitmap. This will be 0 if the function fails or the image does not exist, and the handle of the specified color channel otherwise.

Remarks:

Channels are only relevant to images with separate colour channels (RGB or CMY, RGBA or CMYK). If you use `GetChannel` on a single channel image, the return value is the handle to the bitmap. I.e. equivalent to the `TuneImage(0,0,0,0,0,Handle)` or `SetCurrentBitmap(Image$,FULL,Handle)` commands. `GetChannel` makes a new bitmap in memory: its up to you to free this once you are done with the `FreeChannel` command.

Example:

```
GetRGBchannels:
    {shows the HANDLE of each channel in the current image.}
    GetChannel(RED,RedHandle)
    GetChannel(GREEN,GreenHandle)
    GetChannel(BLUE,BlueHandle)
    DrawText(10,30,"RedHandle")
    DrawNumber(120,30,RedHandle)
    DrawText(10,50,"GreenHandle")
    DrawNumber(120,50,GreenHandle)
    DrawText(10,70,"BlueHandle")
    DrawNumber(120,70,BlueHandle)

    Goto Wait_for_Input
```

Related Commands:

[CombineChannels](#) `FreeChannel` [ReplaceChannel](#)

GetThemeStats

geoPIXCL command. Once a theme image has been created by the [PPDClassify](#) or [MLHClassify](#) command, you can export a report of the theme pixels on a per theme basis. The theme image has to be loaded into the geoPIXCL image list.

Syntax: `GetThemeStats(Handle,THMfile$, Result)`

Parameters:

Handle The handle of the image. Use `SetCurrentBitmap` to get this value.
THMFile\$ The name of the disk file to be created. It should ideally have the file extension THM, though this is not mandatory.
Result 1 if the operation was successful, otherwise 0.

Remarks:

A THM file has the format shown below.

```
Input theme file = C:\PixCLTools\SurimiTest\specks.thm
Number of Lines = 1200
Number of Pixels = 1200
Total Pixels = 1440000
Percentage Total may not reach exactly 100.00 due to rounding.
Zero counts are not included in the report.
Theme Number      #Pixels      Percentage Area

Unclassified 1433183      99.5266
# 1           6817         0.473403
Percentage Total = 100.0
```

Related Commands:

[PPDClassify](#) [MLHClassify](#)

InvertImage

Invert the colors of the target bitmap in the PiXCL image list.

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [InvertImage\(Result\)](#)

Parameters:

[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

```
Inversion:
    InvertImage(Res)
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
    Goto Wait_for_Input
```

Related Command:

[InvertRectangle](#)

LinearEnhanceImage

One of the most useful enhancement commands for a target bitmap in the PiXCL image list is the automatic linear enhance. This reads the initial bitmap of the current bitmap data to calculate the mean and standard deviation, for each channel, then applies a linear stretch plus/minus two standard deviations from the mean (subject to clipping at 0 and 255).

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `LinearEnhanceImage(Mean&,StdDeviation&,Result)`

Parameters:

Mean&, StdDeviation& The desired mean and standard deviation for the resulting image. The default values are 128.0 and 35.0. If you set either value to 0.0, the default value is used.

Result Non-zero if the process was successful, otherwise 0.

Remarks:

You can obtain the mean and standard deviations for a PiXCL image list bitmap with the [ReportHistogramStats](#) command, and calculate the $y = Bx + C$ enhancement coefficients (**A** is set to 0.0) for input into the [CalibrateImage](#) command as follows.

```
B& = desired_StdDev& / current_StdDev&
C& = desired_Mean& - B& * current_Mean&
```

[LinearEnhanceImage](#) currently takes the channel average mean and standard deviation to calculate A, B and C internally, and then runs the [CalibrateImage](#) function. Hence, the actual resulting mean and standard deviation will be approximately what you specified. If the original data is skewed to low or high values, the resulting mean and standard deviation will also be affected. This is normal.

Example:

```
Linear:
  LoadBitmap(ImageFile$,FULL)
  LinearEnhanceImage(128.0, 35.0, Res)
  If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
  Goto Wait_for_Input
```

Related Commands:

[Histogram](#) [UpdateHistogram](#) [ShowHistogram](#) [ReportHistogramStats](#)

MakeNDVIimage

geoPIXCL command. Using two single 8-bits per channel images of the same dimensions and geographically, temporally and/or spectrally related to each other, a Normalized Difference Vegetation Index image can be created.

Algorithm Description

The techniques for monitoring drought from satellite were developed by NOAA National Environmental Satellite Data and Information Service. The NOAA Global Vegetation Index is issued weekly as a database in real time for each 16 x 16 km square, between 75° N to 55° S latitude. It is available from the NOAA office in Washington D.C. This database product was designed through spatial and temporal sampling, then mapping and calculating the Normalized Difference Vegetation Index (NDVI) from the sun's electromagnetic reflectance in the visible and near infrared spectral bands acquired by the AVHRR satellite.

The NDVI concept is based on the properties of green vegetation to reflect solar radiation in two spectral bands sampled by the AVHRR sensor: visible, 0.55-0.68µm (Channel 1), and near-infrared, 0.725-1.10µm (Channel 2). The presence of chlorophyll pigment in green vegetation and the leaf scattering mechanisms cause low spectral reflectance in Channel 1 and high spectral reflectance in Channel 2.

Reflectance values change in the opposite direction if the vegetation is under climatic or growth pattern stress. Thus, the difference between the values of these two channels was selected by NOAA as a measure of the degree of vegetation "greenness".

The general formula used is

$$\text{NDVI} = (\text{Ch2} - \text{Ch1}) / (\text{Ch2} + \text{Ch1})$$

or for this command

$$\text{Pxy-out} = \text{gainC} * ((\text{gainA} * \text{ch1} + \text{offsetA}) - (\text{gainB} * \text{ch2} + \text{offsetB})) / ((\text{gainA} * \text{ch1} + \text{offsetA}) + (\text{gainB} * \text{ch2} + \text{offsetB})) + \text{offsetC}$$

where ch1, ch2 == Pxy-in

Inspection of this formula, using 8-bit files as the inputs Ch1 and Ch2 show that the NDVI can range from a value of -1.0 to + 1.0. In practice, NDVI is a good measure of the density and vigor of green vegetation. For rocks, desert and bare soil, the NDVI is always near zero, while for lush vegetation, the NDVI calculated from satellite measurements is close to its practical maximum of about 0.6.

Syntax: `MakeNDVIimage(HandleA,HandleB, NDVIimage$, gainA&, offsetA&, gainB&, offsetB&, gainC&, offsetC&, Result)`

Parameters:

| | |
|---------------------------------------|--|
| <code>HandleA, HandleB</code> | The handles of the input channels loaded into the geoPiXCL image list. |
| <code>NDVIimage\$</code> | The name of the resultinv image that gets created in the image list. |
| <code>gainA&, offsetA&</code> | The calibration values for channel A. The default values are 1.0 and 0.0. |
| <code>gainB&, offsetB&</code> | The calibration values for channel B. The default values are 1.0 and 0.0. |
| <code>gainC&, offsetC&</code> | The calibration values for the output image. The default values are 350.0 and 0.5. |
| <code>Result</code> | 1 if the operation succeeded, otherwise, 0. |

Related Commands:

[ComputeImage](#)

MakeScattergram

geoPiXCL command. Using two single 8-bit channel images of the same dimensions and geographically, temporally and/or spectrally related to each other, a scattergram or cross plot of the image data is created.

A third single channel image, typically the result of a classification, can be read to act as a mask for the scattergram creation. The resulting image is a 256 pixel square, 256 color bitmap, and is written to a memory bitmap, or automatically written to disk, where it can be used by other plotting moduled or commands.

Scattergrams are useful in evaluating the overall relationship of one wavelength channel to another, or the relationship of spectral classes within a pair of images, using a thrid mask channel.

Syntax: *MakeScattergram(Xhandle, Yhandle, MaskHandle, Mask\$, SamplePercent, ScatterBitmap\$, Result)*

Parameters:

| | |
|-------------------------|--|
| <i>Xhandle, Yhandle</i> | The handles of the images loaded into the geoPiXCL image list. These handles are returned by SetCurrentBitmap and TuneImage . |
| <i>MaskHandle</i> | The handle of an optional mask image. Set this to 0 if a mask is not used. |
| <i>Mask\$</i> | A string that defines the mask entries to be used. A set of image mask index values can specified, as follows: <ul style="list-style-type: none">- a single index number in the range 0 - 254: only mask pixels with this index will be counted.- index 255: count all values. Scatterplot color = theme color, multi-class pixels are white (index 255).- a set of values, ',' delimited, up to 32. e.g. the arg could be "1,3,4,5,8" ... no spaces.- a range of values. e.g. 1-8- a set and range e.g. 1,2,5-8 |
| <i>SamplePercent</i> | The sample rate for the scattergram generation. Acceptable values are 100, 50, 33 and 25. Any other values default to 100. |
| <i>ScatterBitmap\$</i> | The name of the scattergram always created on disk and loaded into the geoPiXCL image list. |
| <i>Result</i> | 0 if the operation fails, or the handle of the bitmap that is loaded into the image list. |

Remarks:

The operation will fail if the bitmaps are not 8 bits per pixel, and/or are not the same dimensions.

The scattergam image for X and Y channels with no mask has a white background with dark green scatter pixels. Masked scattergrams have black backgrounds with scatter pixels in the standard geoPiXCL class sequence.

Related Commands:

[Histogram](#) [ShowHistogram](#)

MakeSpectralSignatureFiles

geoPIXCL command. Once training areas have been created, a spectral signature is usually required. The spectral signature generator function can process up to 32 training areas and eight input bands, to make a set of ASCII SIG files.

The function differs from other functions, in that file names are passed, rather than a set of image handles that are loaded into memory.

The sets of TRG names and Imagenames are "]" delimited. A single SIG file is created for each TRG file, using the imagenames supplied. All imagenames must be

- 8 bits per pixel hence BMP and TIF are the supported formats. Other formats are usable.
- same #lines, #pixels.

Images are NOT loaded into memory, and instead we read in a line at a time, as needed, while the signatures are generated. This is because a training area is generally a small region or regions of the whole file.

A spectral signature is generalized parameter. There is no difference in a signature created for a classification process or a principal component process. Hence, any new process that needs a signature can use this tool.

Syntax: *MakeSpectralSignatureFiles(TRGfileList\$, TRGcount, ImageList\$, ImageCount, ONDISK|MEMORY, Result)*

Parameters:

| | |
|----------------------|---|
| <i>TRGfileList\$</i> | A "[" delimited list of input TRG files. Filenames can include 'space' characters in the path. |
| <i>TRGcount</i> | The number of filenames in <i>TRGfileList\$</i> . |
| <i>ImageList\$</i> | A "[" delimited list of not less than two input images. These images have to be the same dimensions as the mask image from which the TRG files were generated. |
| <i>ImageCount</i> | The number of filenames in <i>ImageList\$</i> . |
| ONDISK | For 8-bit BMP files in <i>ImageList\$</i> only, the images are accessed as needed on disk, to create the spectral signature files. |
| MEMORY | For any 8-bit images, the whole image is loaded into memory and the relevant pixels accessed to create the spectral signature files. Images are deleted from memory once the operation is complete. |
| <i>Result</i> | 0 if the operation failed, otherwise the number of SIG files created. |

Remarks:

Spectral signature files contain the names of the images that were used, the size and mean pixel value of the training area, and the covariance matrix for the set of input images.

Related Commands:

[ExtractTRGAreaFiles](#)

MartinTaylorMapping

geoPIXCL command. Another “sort-of” Principal Component inverse function that provides a useful enhancement.

Syntax: *MartinTaylorMapping(Handle, MartinTaylorImage\$, Result)*

Parameters:

Handle The handle of a current 24-bit image in the PiXCL list. This is typically a principal components composite.

MartinTaylorImage\$ The name of the image to created in the geoPiXCL image list.

Result 1 if the operation was successful, otheriwise 0.

Related Commands:

[DecorrelStretchImage](#) [PCEnhanceImage](#)

MLHClassify

geoPIXCL command. To perform a maximum likelihood classification of up to 32 input images, a “|” delimited list of the spectral signature files is created and passed to the [MLHClassify](#) command.

Syntax: [MLHClassify](#)(*SigList\$, ThemeImage\$, PALfile\$, Result*)

Parameters:

| | |
|------------------------------|---|
| SigList\$ | A list of the SIG files that were previously created from a corresponding set of TRG files. |
| ThemeImage\$ | The resulting theme image created in the geoPIXCL image list. It is an 8 bit paletted image the same dimensions as the input image set. This theme file can be transparently overlaid on to other images with the OverlayImage command. |
| PALFile\$ | Set to an empty string "" to use the default geoPIXCL theme (class) colour scheme, or specify a valid PAL or PBL file to set a custom theme colour palette. |
| Result | 1 if the operation was succesful, otherwise 0. |

Related Commands:

[ExtractTRGAreaFiles](#) [OverlayImage](#) [MakeSpectralSignatureFiles](#) [ConvertPALfile](#)

ModeFilterImage

geoPIXCL command. A mode filter takes the value of the mode pixel in the neighbourhood surrounding a center pixel and replaces the center pixel with that value. It is generally used to clean up small areas in 8 bit theme images, but can also be used on an 8 bit grey scale image if desired. For the cases of pixels along the edge of an image, neighbourhood pixels effectively outside the image are assumed to be the same as the center pixel.

Syntax: `ModeFilterImage(ImageName$, ThemeValue, MinMode, Xsize, Ysize, Result)`

Parameters:

| | |
|---------------------|---|
| <i>ImageName\$</i> | An 8 bit theme image loaded in the geoPIXCL image list. If the image is not in the list Result returns 0. |
| <i>ThemeValue</i> | The theme pixel value to be processed. Only pixels of this value are processed and all other pixels are left unchanged. |
| <i>MinMode</i> | The minimum mode pixel count that will permit the center pixel to be changed. For example, to remove a single theme pixel surrounded by pixels of the same colour, the neighbourhood should be 3x3, and <i>MinMode</i> should be 8. |
| <i>Xsize, Ysize</i> | The size of the neighbourhood, in odd numbers, to a maximum of 7. <i>Xsize</i> and <i>Ysize</i> do not have to be the same value. |
| <i>Result</i> | 1 if the operation was successful, othersize 0. |

Related Commands:

[Filter5x5](#) [Filter15X15](#)

NLEnhanceImage

Non-linear Logarithmic, exponential and power enhancement functions can be done with the [NLEnhanceImage](#) command as follows.

Logarithmic enhancement: $P_{out} = Gain * \log K(P_{in}) + Offset$ where $1.00 \leq K \leq 100.00$

Exponential enhancement: $P_{out} = Gain * K^{**}P_{in} + Offset$ where $0.75 \leq K \leq 1.25$

Power enhancement: $P_{out} = Gain * P_{in}^{**}K + Offset$

Syntax: [NLEnhanceImage\(Handle, NewImage\\$, LOG|EXP|PWR, Kfactor&, Gain&, Offset&, MinPixel, MaxPixel, Result\)](#)

Parameters:

| | |
|--|---|
| Handle | Image handle returned from TuneImage . |
| NewImage\$ | A name if you want a new result image to be created in the PiXCL list. To process the existing image, set this to "" (a null string). |
| LOG | Do a logarithmic enhancement. Kfactor& = 2.00 is good starting point. |
| EXP | Do an exponential enhancement. Kfactor& = 1.022 is good starting point. |
| PWR | Do a power function enhancement. Use Kfactor& = 0.5 for squareroot. |
| KFactor& | The factor for the Log, Exponent or Power function. |
| Gain&, Offset& | Set a gain and offset factor to scale the result pixel values. Set these to 1.0 and 0.0 results in automatic gain and offset calculation to scale the MinPixel , MaxPixel results to 0, 255. Note this does NOT set the gain and offset to 1.0 and 0.0. Use, say, 1.01 and 0.01 if you need these values. |
| MinPixel, MaxPixel | Set the range of values to process. MinPixel cannot equal 0. PiXCL automatically changes 0 values to 1. You will need to examine the image histogram to decide these values. |
| Result | 1 if the operation was successful, otherwise 0. |

Remarks:

Logarithmic enhancement increases contrast in the lower pixel values, while exponential enhancement increases contrast in the higher pixel values. Power functions (e.g. 0.5) tend to smooth the image.

In general, a good starting point for non-linear enhancement processes is to use the default gain and offset values.

Related Commands:

[CalibrateImage](#) [EqualizeImage](#) [NormalizeImage](#) [NormalizeImageRange](#) [Histogram](#) [ShowHistogram](#)

NormalizeImage

[NormalizeImage](#) creates a histogram from the pixel values in the current bitmap and normalizes the range of values in the histogram. The histogram is then remapped to the current bitmap. The histogram creation is not reported in the progress reporting since it does not take long to create.

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [NormalizeImage\(Result\)](#)

Parameters:

[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

```
Normalizer:
  NormalizeImage(Res)
  If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
  Goto Wait_for_Input
```

Related Command:

[EqualizeImage](#) [NormalizeImageRange](#)

NormalizeImageRange

[NormalizeImageRange](#) creates a histogram from the pixel values in the current bitmap and normalizes a specified range of values in the histogram. The histogram is then remapped to the current bitmap. The histogram creation is not reported in the progress reporting since it does not take long to create.

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [NormalizeImageRange](#)(*Min*, *Max*, *Result*)

Parameters:

Min, *Max* The working range, 0 - 255. *Min* must be less than *Max*.
Result Non-zero if the process was successful, otherwise 0.

Example:

```
NormalizerRange:  
    NormalizeImageRange(10,170,Res)  
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)  
    Goto Wait_for_Input
```

Related Command:

[EqualizeImage](#) [NormalizeImage](#)

OverlayImage

There are situations where you will want to transparently overlay one bitmap loaded in memory with a another bitmap of the same dimensions. For example, the base or background image is a 24 bit color image, and the overlay image is an 8 bit color image made of lines and regions of various colors, on a black background. Only the non-black information is to be overlaid on to the base image.

This command provides a means to do a transparent overlay of a foreground image on to a background image, such that only foreground pixels that are not the transparent color are written. The background image must already be in memory, and be the current bitmap, using one of the [DrawBitmap](#) command set. [OverlayImage](#) works with all the supported bitmap formats. In most cases, the background image will be a 24 bit image e.g. BMP, JPEG, TIF, while the foreground image can be either 8 bit or 24 bit. Background and foreground bitmaps do not have to be the same image type on disk.

You can save the result of the overlay operation with the [SaveBitmap](#) command.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [OverlayImage](#)(*ForegroundImage\$, TrRed, TrGreen, TrBlue, Result*)

Parameters:

| | |
|-----------------------------------|---|
| ForegroundImage\$ | Name of the foreground image already in memory. |
| TrRed | The red transparency color in range 0-255. |
| TrGreen | The green transparency color in range 0-255. |
| TrBlue | The blue transparency color in range 0-255. |
| Result | 1 if the operation was successful, otherwise 0. |

Remarks:

The red, green, blue color value defines the transparency color in the overlay image. These numbers must be in the range 0 - 255, and numbers outside this range are set to 0. You can use the [ChooseColor](#) command to get an R,G,B triplet.

Example:

This code fragment has an image already loaded ([ImageFile\\$](#)), loads the overlay image, sets the current bitmap with the [DrawBitmap](#) command, then performs the overlay function. Finally, the updated bitmap is displayed with another [DrawBitmap](#) command.

```
OverlayThemes:  
  LoadBitmap(Image9$)  
  DrawBitMap(23,23,ImageFile$)  
  OverlayImage(Image9$, 0,0,0, Res)  
  DrawBitMap(23,23,ImageFile$)  
  Goto Wait_for_Input
```

Related Commands:

[DrawTrBitmap](#) , [DrawSizedTrBitmap](#) , [SaveBitmap](#)

PCEnhanceImage

geoPIXCL command. Based on a previously created set of spectral signature (.SIG) files, create a set of principal component images.

The signature file that has been previously generated from the same set of images is passed to the function. Note the image names are listed in the SIG file. The images are loaded and the desired number of PC images created.

The Principal Components process is a statistical colour space conversion process, and requires two passes. The first generates the correction parameters, based on the signature covariance matrix, and the second pass performs the actual conversion to the PCA colour space.

The result is blue-yellow, red-green image when a 24-bit colour composite is created later. This is not the same as a RGB-IHS/VHS conversion, as these process are purely a fixed conversion and makes no assumptions about the input channel data space.

An ASCII file, extension .PCM, containing the transformation matrix is created during the process.

Syntax: `PCEnhanceImage(SigFile$, BaseFile$, Components, MODE_token, Result)`

Parameters:

| | |
|-------------------|---|
| <i>SigList\$</i> | A list of the SIG files that were previously created from a corresponding set of TRG files. |
| <i>BaseFile\$</i> | The name that is used to create the output channel names. The last four characters of the name are replaced with PCAn, where 'n' is the component number starting from 1. For example, if the input is "brsfnd1.bmp", the first PC file will be "brsfpca1.bmp". Note that only BMP and TIF/geoTIF formats are supported at present. |
| <i>Components</i> | The number of principal components channels to be created. This is <= to the number of input bands specified in the SIG file. |
| <i>MODE_Token</i> | Sets the processing mode. <code>LISTADD</code> creates the output files in the image list. <code>DISKONLY</code> creates the output files on disk. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[ExtractTRGAreaFiles](#) [OverlayImage](#) [MakeSpectralSignatureFiles](#) [ConvertPALfile](#)

PPDClassify

geoPIXCL command. To perform a parallelepiped classification (essentially a simple segmentation) of up to 8 input images, a class range is created and passed to the **PPDClassify** command. Up to 32 classes can be created in one pass.

Syntax: `PPDClassify(Handle[Index],ThemeImage$,RangeData$, PALfile$, NumClasses, Result)`

Parameters:

| | |
|----------------------|--|
| <i>Handle[Index]</i> | An array of up to 8 image handles for the input channels. Set unused handles to 0. Index is the start index in the array for the handles. This will usually be 0. You must create the array prior to running this command. |
| <i>ThemeImage\$</i> | The resulting theme image created in the geoPIXCL image list. It is an 8 bit paletted image the same dimensions as the input image set. This theme file can be transparently overlaid on to other images with the OverlayImage command. |
| <i>RangeData\$</i> | The name of a data file, or the data itself, read from a file, or generated within the geoPIXCL application. Range data is in the form of a vector of class low and high pixel values for each input channel sequentially. A range has the format of one class entry per line, space delimited, with a cr-lf terminator. For example, a two class range for an RGB image might be 10 26 13 35 18 44 crlf for class #1 39 67 51 88 77 93 crlf for class #2 Ideally, the class ranges should not overlap, otherwise the results will be meaningless. In addition, there is no class priority, so that a pixel assigned to class #1 that subsequently is deemed to be in class #2 will be assigned to class #2. |
| <i>PALFile\$</i> | Set to an empty string "" to use the default geoPIXCL theme (class) colour scheme, or specify a valid PAL or PBL file to set a custom theme colour palette. |
| <i>NumClasses</i> | The number of classes to create, up to 32. If $0 \leq \text{NumClasses} > 32$, the function will fail, and no class image will be created. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[ExtractTRGAreaFiles](#) [OverlayImage](#) [MakeSpectralSignatureFiles](#) [ConvertPALfile](#)

ReadLANfileHeader

geoPIXCL command. This command reads the 128-byte header of Erdas [™] LAN and GIS files, and extracts the image parameters.

Syntax: `ReadLANfileHeader(LANImage$, Hdword$, Ipack, Bands, Icols, Irows, Xstart, Ystart, MapType, Nclass, IauType, Acre&, Xmap&, Ymap&, Xcell&, Ycell&)`

Parameters:

| | |
|-------------------------------|---|
| <i>LANImage\$</i> | The LAN/GIS image to read. |
| <i>Hdword\$</i> | A 6-character string containing "HEAD74", or "HEADER" for pre-v7.4 files. |
| <i>Ipack</i> | The pack type of the image data. 0 = 8-bit, 1 = 4-bit, 2 = 16-bit. |
| <i>Bands</i> | The number of bands or channel per line. For GIS files, this is always 1. For LAN files, this is commonly 1 or 3. |
| <i>Icols</i> | The number of columns or pixels per line. |
| <i>Irows</i> | The number of rows or lines in the image. |
| <i>Xstart</i> | The database x-coordinate of the first pixel in the image i.e. top left corner. |
| <i>Ystart</i> | The database y-coordinate of the first pixel in the image i.e. top left corner. |
| <i>MapType</i> | An integer indicating the type of map projection used. |
| <i>Nclass</i> | The number of classes in the dataset. |
| <i>IauType</i> | The unit of area of each pixel. 0 = none 1 = acre 2 = hectare 3 = other |
| <i>Xmap&, Ymap&</i> | Real numbers that gives the X and Y map coordinates of the top left corner pixel. |
| <i>Xcell&, Ycell&</i> | Real numbers that gives the X and Y size of each cell. |

Related Commands:

[EditLANfileHeaderBox](#) [WriteLANfileHeader](#)

ReadSPOTData

geoPIXCL Command. SPOT Image Corporation imagery can be read directly with a command, into the geoPIXCL image list. Level 1A/1B panchromatic and multispectral data sets can be read, either entirely, or a partial rectangle.

Syntax: `ReadSPOTData(SPOTfile$, OutFile$, InLines, InPixels, OutLines, OutPixels, StartLine, StartPixel, MODE_token, INVERT_token, Result)`

Parameters:

SPOTfile\$ The file to be read into the geoPIXCL image list, or read and converted into a bitmap file on disk, if *OutFile\$* is not null.

OutFile\$ The name of the output disk file.

InLines, InPixels The number of lines and pixels read from the SPOT header file.

OutLines, OutPixels The number of lines and pixels written to the output file or bitmap in memory. The same as *InLines, InPixels* when the whole image is to be read.

StartLine, StartPixel The image coordinate where data reading starts.

MODE_token

| | |
|------|-----------------------|
| "P" | panchromatic format |
| "XS" | multi-spectral format |
| "15" | SPOTview 1.5 format |
| "40" | SPOTView 4.0 format |

INVERT_token "STD|INVERT" indicates whether the image is inverted. INVERT is required for image loads into the geoPIXCL image list.

Result 1 if the operation was successful, otherwise 0.

Related Commands:

[ReadSPOTfileHeader](#)

ReadSPOTfileHeader

geoPIXCL command. Imagery supplied by the SPOT Image Corporation on CD-ROM include a set of files. One, the image header file, contains the information of the image parameters. This command reads that file and extracts information on the number of lines and pixels and imagery types.

Syntax: `ReadSPOTfileHeader(DATfile$, Lines, Pixels, Interleaving$, Spectralband$, Bandtype$)`

Parameters:

| | |
|-----------------------|---|
| <i>DATfile\$</i> | The header file LEAD*.dat to be read. |
| <i>Lines, Pixels</i> | The number of lines and pixels of the SPOT image defined in <i>DATfile\$</i> . |
| <i>Interleaving\$</i> | The interleaving type string. E.g. BIL. |
| <i>Spectralband\$</i> | The spectral band identifier. PAN = panchromatic mode, XS1 = multispectral band1. |
| <i>Bandtype\$</i> | Indicates the sensor origin. 1A or 1B . |

Related Commands:

[ReadSPOTData](#)

RemapImage

With this command you can remap pixel colors, using an imported ASCII text palette file. [RemapImage](#) is a direct interface to pixel remapping permitting any remapping of source pixel values to destination pixel values. Please note that **this is NOT a palette remapping** function, and it does not work on the basis of manipulating color palette index values.

[RemapImage](#) is used to change the colors of an entire image, and is related to the brightness and contrast manipulation functions like [TuneImage](#), [EqualizeImage](#) and [NormalizeImage](#). See the **Remarks** section below for further explanation. Palette files can be created with the [CreatePALfile](#) command.

Color Support: Indexed, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [RemapImage](#)(*ColorArrayFile* \$, *Result*)

Parameters:

[ColorArrayFile](#) \$ The color array filename.

[Result](#) 1 if the operation succeeded otherwise 0.

Palette File format.

This ASCII text file consists of three lines of identifiers followed by 256 RGB array entries. The ! and following text are shown here as comments only and are not part of the format. Entries are space or TAB separated and have a carriage-return / linefeed at the end of each line.

```
PiXCL-PAL      ! identifier. JASC-PAL (from Paint Shop Pro 4,5,6) is also recognized.
0100            ! version number
256            ! number of entries
0 0 0           ! array entry #1
255 0 0        ! array entry #2
0 255 0
0 0 255
255 255 0
255 0 255
0 255 255
255 128 0
192 192 192
0 128 0
0 0 128
255 0 128
...
0 0 0           ! palette entry #256
```

It is essential that you understand that the array values are applied to each color channel individually. The read array values are applied to ALL pixel values in the Red channel. 8 bit paletted images are considered in effect to be 24 bit images.

Consider two mapping arrays as shown below.

```
0 0 0            255 255 255
1 1 1            254 254 254
2 2 2            253 253 253
. . .            . . .
255 255 255      0 0 0
```

If you apply the left mapping array to an image, there will be no change to the target image, but if you apply the right hand array, the image colors will be inverted.

Hence, with [RemapImage](#) you can, for example, perform

1. non-linear enhancements by creating new RGB mapping arrays that replace the linear 0 - 255 standard values.
2. piece-wise linear enhancements that remap ranges of values to different functions.
3. Map ranges of pixel values to the same color. This is sometimes referred to as image classification.

Let us consider case #3 above. For example, suppose that you want all pixels that fall in the ranges **Red[43-56]** , **Green[68-75]** , and **Blue[78-85]** to be displayed as red (255,0,0). Take the standard linear 0 - 255 array file, and in the red channel, set all values in the range 43 - 56 to 255, in the green channel, set all values in the range 68 - 75 to 0, and in the blue channel, set all values in the range 78 - 85 to 0.

When the mapping is applied, only pixels in the desired range will be set to red, while the the rest of the image will be unchanged. To reset the colors, you have to reload the original image: applying a standard linear 0 - 255 mapping array will not reset the original colors because the image data has been changed.

Array files can be created with a PiXCL script, entered into a file by hand, or created by other programs.

Related Commands:

[CreatePALfile](#) [TuneImage](#) [EqualizeImage](#) [NormalizeImage](#)

RemapTheme

geoPIXCL command. Thematic images (8 bits / pixel) can contain various classes which sometimes need to be changed. With the RemapTheme command, you specify an input theme that is to be turned into a target theme. Within the theme image, the pixel values are changed, not the image colour map.

Syntax: `RemapTheme(ThemeImage$, InputTheme, TargetTheme, Result)`

Parameters:

| | |
|---------------------------|--|
| <code>ThemeImage\$</code> | A theme image loaded into the geoPiXCL image list. |
| <code>InputTheme</code> | The theme number to be changed. |
| <code>TargetTheme</code> | The theme number (i.e. colour) to which the <code>InputTheme</code> will be converted. |
| <code>Result</code> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[ModeFilterImage](#) [CombineThemes](#)

ReplaceChannel

This command is used to experiment with multiple channel images, to see the effects of replacing individual channels in color composites.

Syntax: `ReplaceChannel(RGBHandle, GRAYhandle, Channel, Result)`

Parameters:

| | |
|-------------------|--|
| <i>RGBHandle</i> | Handle of a 24 bit image already in memory. |
| <i>GRAYhandle</i> | Handle of a 8 bit image already in memory. |
| <i>Channel</i> | The 0-indexed channel in the 24 bit image that is to be replaced i.e. 0 = Red , 1 = Green , 2 = Blue . |
| <i>Result</i> | 0 if the operation fails, otherwise is the non-zero handle of the new image. |

Example:

This example displays a bitmap (previously loaded), gets the handle, then replaces the blue channel with another image loaded in memory, and displays the result.

Replacer:

```
DrawBitmap(23,23,Image8$)
DrawText(0,0,"RGB24 handle= ")
DrawNumber(135,0,Band8Handle)
TuneImage(0,0,0,0,0,0,Band8Handle)
DrawNumber(230,0,Band5Handle)
ReplaceChannel(Band8Handle,Band5Handle,2,Res)
If Res = 0
    MessageBox(OK,1,INFORMATION,
    "Replace Channel failed","",Res)
Else
    DrawBitmap(23,23,Image8$)
Endif

Goto Wait_for_Input
```

Related Commands:

[CombineChannels](#), [SaveBitmap](#)

ResampleImage

[ResampleImage](#) provides a means to change the size and aspect ratio of an image by either bi-cubic, bi-linear or nearest neighbor methods. The resampling process is especially useful when you need to process 24 bit images to adjust the dimensions (i.e. the aspect ratio), or when you have images from sources that are of different resolution, and you need to overlay one image with another.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [ResampleImage\(Pixels, Lines, NEAREST | BILINEAR | BICUBIC, Result\)](#)

Parameters:

| | |
|-------------------------------|--|
| Pixels, Lines | The desired dimensions of the resized image. |
| NEAREST | Nearest Neighbor method, low quality but fast. |
| BILINEAR | Bi-linear method, high quality but slower. |
| BICUBIC | Bi-cubic method, best quality, slowest. |
| Result | Non-zero if the process was successful, otherwise 0. |

Example:

Resampler:

```
ResampleImage(200,200,NEAREST,Res)
If Res <> 0
    DrawBackGround
    DrawBitMap(50,50,ImageFile$)
Endif
Goto Wait_for_Input
```

Related Commands:

[CropImage](#) [ResizeImage](#)

ResizeImage

[ResizeImage](#) provides a means to change the size and aspect ratio of the current bitmap image using a bi-linear method.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [ResampleImage](#)(*Pixels, Lines, Result*)

Parameters:

Pixels, Lines The desired dimensions of the resized image.

Result Non-zero if the process was successful, otherwise 0.

Example:

Resizer:

```
ResizeImage(200,200,Res)
If Res <> 0
  DrawBackGround
  DrawBitMap(50,50,ImageFile$)
Endif
Goto Wait_for_Input
```

Related Commands:

[CropImage](#) [ResampleImage](#)

RotateImage, RotateImageBkg

An image can be rotated around its center pixel by a specified angle and the resulting image enclosed in a rectangular bitmap with the background set to black. If you want the background colour to be another color, use the [RotateImageBkg](#) command

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [RotateImage\(Amount,Result\)](#)
[RotateImageBkg\(Amount&,resample_Mode,bkgR, bkgG, bkgB, Result\)](#)

Parameters:

| | |
|----------------------------------|--|
| Amount | A number in degrees between 1 and 360. Larger numbers are automatically converted to the modulus 360 number. Negative numbers are permitted. |
| Amount& | A floating point number in degrees between 1.0 and 360.0 Negative numbers are permitted. |
| resample_Mode | NEAREST Nearest Neighbor method, low quality but fast. BILINEAR Bi-linear method, high quality but slower. BICUBIC Bi-cubic method, best quality, slowest. |
| bkgR, bkgG, bkgB | The CURRENT background colour for the rotated image. This will usually be the default black (0,0,0), and is the colour that a previous call to UseBrush replaces. |
| Result | Non-zero if the process was successful, otherwise 0. |

Example:

ImageRotation:

```
RotateImage(-10,Res)
If Res <> 0 Then DrawBitmap(10,10,ImageFile$)
Goto Wait_for_Input
```

SavingFile:

```
WaitInput(1)
Image$ = "..\images\brsfc754.bmp"
DrawBackground
DrawBitmap(20,40,Image$)
Rotate& = -170.0
UseBrush(SOLID,160,255,128)
UsePen(SOLID,1,255,0,0)
RotateImageBkg(Rotate&,BICUBIC,0,0,0,Res)
WaitInput(250)
GetListBitmapDim(Image$,Lines,Pixels,Bits)
Lines += 41 Pixels += 21
DrawRectangle(19,39,Pixels,Lines)
DrawBitmap(20,40,Image$)
SaveImage$ = SourceDir$ + "\savetest.bmp"
SaveBitmap(SaveImage$,Res)
FreeBitmapAll
Goto Wait_for_Input
```

Related Commands:

[GeoCorrectImage](#) [GeoTranslateImage](#) [RemapImage](#) [RotateImageExt](#)

RotateImageExt

An image can be rotated around a selected pixel by a specified angle, while the resulting image dimensions do not change. For example, if the selected pixel is in the top left quadrant of the image, you will lose some portions of the original image data in the rotation. If you want the background colour to be another color, a subsequent call to RemapImage should be used.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `RotateImageExt(rx,ry,Amount,mode_TOKEN,Result)`

Parameters:

| | |
|-------------------|--|
| <i>rx,ry</i> | The selected point that is used for the rotation. |
| <i>Amount</i> | A number in degrees between 1 and 360. Larger numbers are automatically converted to the modulus 360 number. Negative numbers are permitted. |
| <i>Mode_TOKEN</i> | NEAREST: nearest neighbour resampling BILINEAR: bilinear interpolation resampling BICUBIC: bicubic interpolation resampling |
| <i>Result</i> | Non-zero if the process was successful, otherwise 0. |

Example:

```
ImageRotation:
  RotateImageExt (rx, ry, -10, BICUBIC, Res)
  If Res <> 0 Then DrawBitmap(10,10,ImageFile$)
  Goto Wait_for_Input
```

Related Commands:

[GeoCorrectImage](#) [GeoTranslateImage](#) [RemapImage](#) [RotateImage](#)

ScatterPixels

This function processes an image to give an effect rather like viewing through frosted glass or glass with raindrops on it. The higher the scatter amount the further apart the pixels are scattered.

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `ScatterPixels(Amount,Result)`

Parameters:

Amount A number between 1 and 99. Larger and negative numbers are automatically converted to the absolute modulus 99 number.

Result Non-zero if the process was successful, otherwise 0.

Example:

```
DisplacePixels:
    ScatterPixels(20,Res)
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
    Goto Wait_for_Input
```

ScaleCropImage

[ScaleCropImage](#) extracts a rectangular section of the current bitmap, discards the original bitmap and scales the new section as the current bitmap.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [ScaleCropImage\(x1,y1,x2,y2,Scale&,Result\)](#)

Parameters:

[x1,y1,x2,y2](#) The coordinates in the image that define the corners of the desired region.

[Scale&](#) A value equal to or greater than 1.0. If a [Scale&](#) less than 1.0 is specified, the command does nothing. Note that [Scale&](#) is a divisor, and the resulting image will be smaller than the coordinate rectangle specified. If you need to make the image larger, use a subsequent [ResampleImage](#) command.

[Result](#) Non-zero if the process was successful, otherwise 0.

Example:

```
ScaleCropper:
    Scale& = 1.31
    ScaleCropImage(20,20,180,220,Scale&, Res)
    If Res <> 0
        DrawBackGround
        DrawBitMap(23,23,ImageFile$)
    Endif
    Goto Wait_for_Input
```

Related Commands:

[CropImage](#) [ResampleImage](#)

ScanBlobObjects

geoPIXCL command. A blob environment contains no blob objects when it is initially created, so the image has to be scanned to locate them.

Syntax: `ScanBlobObjects(BlobEnvID,ImageFile$,ClassValue,ObjectsFound)`

Parameters:

| | |
|---------------------|---|
| <i>BlobEnvID</i> | A number that identifies the blob, returned from CreateBlobEnv . |
| <i>ImageFile\$</i> | An image loaded in the geoPiXCL image list. In the current release, this is required to be 8 bit per pixel. |
| <i>ClassValue</i> | An object class pixel value in the range 1 to 255 that is searched for in <i>ImageFile\$</i> . |
| <i>ObjectsFound</i> | The number of discrete, separable objects found in the image. |

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnv](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#)
[GetBlobCount](#) [SetBlobFilterParams](#)

SetBlobFilterParams

geoPIXCL command. A blob environment has a set of filters that can be turned on and off before measurements are taken. Parameters are applied to ALL blob objects in the Blob Environment. Min/Max default parameters are length = 0.0, imagepixels.0; Size = 0.0, imagepixels.0; edge rejection = off (0.0, 0.0), on (1.0, 0.0).

Syntax: `SetBlobFilterParams(BlobEnvID,LENGTH|SIZE|RESET|EDGEREJECT, MinVal&, MaxVal&, Result)`

Parameters:

| | |
|----------------------------|---|
| BlobEnvID | A number that identifies the blob, returned from CreateBlobEnv . |
| LENGTH | Filter Countable blobs according the min and max length parameters, retaining only those within the range, and setting all others to NonCountable. |
| SIZE | Filter according the min and max size parameters, retaining only those within the range, and setting all others to NonCountable. |
| RESET | Reset all blob objects to Countable state. Length and size parameters are unchanged, but the edge reject parameter is reset. To change these parameters, issue a command with the desired new settings. |
| EDGEREJECT | Set all blobs with pixels on the edge of the image to NonCountable (<i>MinVal&</i> = 1.0) or Countable (<i>MinVal&</i> = 0.0). That is, setting the parameter does actually filter the image. |
| <i>MinVal&</i> | The minimum value for the token, expressed in the current metric used in CreateBlobEnv . For EDGEREJECT mode, 0.0 disables, and 1.0 enables edge rejection. |
| <i>MaxVal&</i> | The maximum value for the token, expressed in the current metric used in CreateBlobEnv . Ignored for EDGEREJECT mode. |
| <i>Result</i> | Non-zero if the process was successful, otherwise 0. |

Related Commands:

[CreateBlobEnv](#) [DrawBlobObjLabels](#) [FreeBlobEnv](#) [FreeBlobEnvAll](#) [ExportBlobObjectData](#) [FilterBlobObjects](#)
[GetBlobCount](#) [ScanBlobObjects](#)

SharpenImage

[SharpenImage](#) uses a matrix to enhance the contrast of areas of an image so that a sharpening effect is given.

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: [SharpenImage\(Amount,Result\)](#)

Parameters:

Amount A number between 1 and 99. Larger and negative numbers are automatically converted to the absolute modulus 99 number.

Result Non-zero if the process was successful, otherwise 0.

Example:

Sharpening:

```
SharpenImage(10,Res)
If Res <> 0 Then DrawBitmap(10,10,ImageFile$)
Goto Wait_for_Input
```


SkewImage

Skews an image vertically or horizontally by the specified number of degrees.

Color Support: Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `SkewImage(Degrees,VERT|HORZ,Result)`

Parameters:

Degrees A number in the range -90 to +90.

VERT|HORZ Vertical or horizontal token.

Result Non-zero if the process was successful, otherwise 0.

Example:

Skewer:

```
SkewImage(45,VERT,Res)
If Res <> 0 Then DrawBitmap(10,10,ImageFile$)
Goto Wait_for_Input
```

TuneImage

TuneImage 'tunes' the brightness and/or contrast of an image, for any or all of the red, green and blue components.

The brightness and contrast percentages must be in the range of -100 and 100. Each element in the array passed to [TuneImage](#) is the percentage used for each color component. The array is in red, green, blue order.

If contrast or brightness must stay the same then set the unwanted parameter to 0..

Color Support: Paletted, Grayscale, RGB24, RGB555, RGB565, RGB32

Syntax: `TuneImage(brightnessR,brightnessG,brightnessB,
contrastR,contrastG,contrastB,Result)`

Parameter:

brightness Array of red, green and blue brightening percentages.

contrast Array of red, green and blue contrast percentages.

Result The image handle if successful, otherwise 0. Image handles are used with other commands, for example, [CalibrateImage](#), [NLEnhanceImage](#).

Example:

```
BrightnessContrastUp:
  Count = 0
  While Count < 10
    TuneImage(10,10,10,5,5,5,Res)
    Count++
    If Res <> 0 Then DrawBitMap(10,10,ImageFile$)
  EndWhile
  Goto Wait_for_Input
```

See Also

[CalibrateImage](#) [NLEnhanceImage](#)

WriteLANfileHeader

geoPIXCL command. This command writes the 128-byte header of Erdas™ LAN and GIS files.

Syntax: *WriteLANfileHeader(LANImage\$, Hdword\$, Ipack, Bands, Icols, Irows, Xstart, Ystart, MapType, Nclass, IauType, Acre&, Xmap&, Ymap&, Xcell&, Ycell&)*

Parameters:

| | |
|-------------------------------|---|
| <i>LANImage\$</i> | The LAN/GIS image to read. |
| <i>Hdword\$</i> | A 6-character string containing "HEAD74", or "HEADER" for pre-v7.4 files. |
| <i>Ipack</i> | The pack type of the image data. 0 = 8-bit, 1 = 4-bit, 2 = 16-bit. |
| <i>Bands</i> | The number of bands or channel per line. For GIS files, this is always 1. For LAN files, this is commonly 1 or 3. |
| <i>Icols</i> | The number of columns or pixels per line. |
| <i>Irows</i> | The number of rows or lines in the image. |
| <i>Xstart</i> | The database x-coordinate of the first pixel in the image i.e. top left corner. |
| <i>Ystart</i> | The database y-coordinate of the first pixel in the image i.e. top left corner. |
| <i>MapType</i> | An integer indicating the type of map projection used. |
| <i>Nclass</i> | The number of classes in the dataset. |
| <i>IauType</i> | The unit of area of each pixel. 0 = none 1 = acre 2 = hectare 3 = other |
| <i>Xmap&, Ymap&</i> | Real numbers that gives the X and Y map coordinates of the top left corner pixel. |
| <i>Xcell&, Ycell&</i> | Real numbers that gives the X and Y size of each cell. |

Related Commands:

[EditLANfileHeaderBox](#) [ReadLANfileHeader](#)

PIXCL Alphabetical Command Reference

This section describes all the commands you can use in a PiXCL 4.4 and 5.0 program. It provides an alphabetical listing of commands with detailed descriptions and sample code.

[AbortShutdown](#) [AboutPiXCL](#) [AboutUser](#) [Acos](#) [AddFont](#) [Ansi](#) [AppWindowHandle](#) [Array](#) [Asin](#) [Atan](#) [Average](#) [AutoProgressBar](#) [Beep](#) [BMWinTitle](#) [Button](#) [ChangeMenuItem](#) [ChangeToolBarBtn](#) [ChooseColor](#) [ChooseFont](#) [Chr](#) [ClearCommPort](#) [ClipboardAppend](#) [ClipboardEmpty](#) [ClipboardGet](#) [ClipboardGetBitmap](#) [ClipboardPut](#) [ClipboardPutBitmap](#) [CloseBitmapWindow](#) [ComboBox](#) [Cos](#) [Cosh](#) [ConvertPALfile](#) [CountBitmapColors](#) [CopyWindowToClipboard](#) [CountCmdLineArgs](#) [Create5x5Filter](#) [CreateBitmap](#) [CreatePALfile](#) [CustomColor](#) [CustomizeToolBar](#)

[DDEExecute](#) [DDEInitiate](#) [DDEPoke](#) [DDERequest](#) [DDETerminate](#)

[DebugMsgBox](#) [DialogBox](#) [DirChange](#) [DirExplore](#) [DirGet](#) [DirGetSystem](#) [DirGetWindows](#) [DirListFiles](#) [DirMake](#) [DirRemove](#) [DiskChange](#) [DlgUnitsToPixels](#) [DragAcceptFile](#) [DrawArc](#) [DrawAnimatedRects](#) [DrawBackground](#) [DrawBackgroundRegion](#) [DrawBitmap](#) [DrawBitmapPoint](#) [DrawBitmapWindow](#) [DrawPreviewBitmap](#) [DrawCaption](#) [DrawChord](#) [DrawEdgeRectangle](#) [DrawEllipse](#) [DrawFlood](#) [DrawFloodExt](#) [DrawFpNumber](#) [DrawFocusRectangle](#) [DrawFrameControl](#) [DrawGrid](#) [DrawIcon](#) [DrawLine](#) [DrawNumber](#) [DrawPie](#) [DrawPoint](#) [DrawPolyCurve](#) [DrawPolygon](#) [DrawPolyLine](#) [DrawRectangle](#) [DrawRoundRectangle](#) [DrawShadeRectangle](#) [DrawShadowFpNumber](#) [DrawShadowNumber](#) [DrawShadowText](#) [DrawShadowTextExt](#) [DrawShellIcon](#) [DrawSizedBitmap](#) [DrawStatusText](#) [DrawStatusWinText](#) [DrawText](#) [DrawTextExt](#) [DrawTrBitmap](#) [DrawTriangle](#) [DrawTrSizedBitmap](#) [DrawVecLabel](#) [DrawVecPoint](#) [DrawVecLine](#) [DrawVecPolygon](#) [DrawZoomedBitmap](#) [DropFileServer](#) [DuplicateImage](#) [EmptyRecycleBin](#) [End](#) [EnlargeImage](#) [EnlargeImageBox](#) [EnumChildWindows](#) [EnumDisplayMonitors](#) [EnumPrinters](#) [EnumWindows](#) [EscCommFunction](#) [ExitWindows](#) [ExpExportHistogram](#) [ExtractListBitmapRect](#)

[FileCopy](#) [FileDecrypt](#) [FileDelete](#) [FileEncrypt](#) [FileExist](#) [FileExtension](#) [FileGet](#) [FileGetDate](#) [FileGetDateExt](#) [FileGetExpandedName](#) [FileGetTempName](#) [FileGetTime](#) [FileGetTimeExt](#) [FileGetSize](#) [FileLZExpand](#) [FileMove](#) [FileName](#) [FilePath](#) [FileRead_ASCII](#) [FileRead_Binary](#) [FileRead_INI](#) [FileRename](#) [FileSaveAs](#) [FileWrite_ASCII](#) [FileWrite_Binary](#) [FileWrite_INI](#) [FindExecutable](#)

[FlashBMWindow](#) [Float](#) [For-Next Loops](#) [FpStr](#) [FpVal](#) [FreeBitmap](#) [FreeBitmapAll](#) [FreeVar](#) [FreeVarAll](#) [GetBackground](#) [GetBitMapDim](#) [GetBMWZoom](#) [GetCmdLine](#) [GetCommPort](#) [GetComputerName](#) [GetCopyDataMsg](#) [GetDialogUnits](#)

[GetDiskSpace](#) [GetDragList](#) [GetEnvString](#) [GetEnvVariable](#) [GetFontFace](#) [GetIPAddress](#) [GetListBitMapDim](#) [GetListBitmapPixel](#) [GetLocalTime](#) [GetMapMode](#) [GetMenuStatus](#) [GetPixel](#) [GetScreenCaps](#) [GetScreenWorkArea](#) [GetSysPowerStatus](#) [GetSystemMetrics](#) [GetSystemTime](#) [GetTempPath](#) [GetTextSpacing](#) [GetTimeZone](#) [GetToolBarBtnStatus](#) [GetViewportExtent](#) [GetViewportOrigin](#) [GetVolumeType](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GlobalMemStatus](#) [Gosub](#) [HexToNum](#) [Histogram](#) [HTMLControl](#) [Hypot](#)

[IDR_Add](#) [Legend](#) [IDR_CloseIris](#) [IDR_DisplayComposition](#) [IDR_GetDataDir](#) [IDR_GetDir](#) [IDR_GetExtensions](#) [IDR_GetLanguage](#) [IDR_GetProgress](#) [IDR_InitProgressTracking](#) [IDR_IsPresent](#) [IDR_Launch](#) [IDR_LaunchModule](#) [IDR_Read_DocFile](#) [IDR_Read_Legend](#) [IDR_Read_Val_DocFile](#) [IDR_Read_Vec_DocFile](#) [IDR_RegisterClient](#) [IDR_SetDataDir](#) [IDR_SetDebugMode](#) [IDR_SetExtensions](#) [IDR_SetProgress](#) [IDR_UnRegisterClient](#) [IDR_Write_DocFile](#)

[If...Then](#) [ImageBox](#) [InfoMenu](#) [InsertListBitmapRect](#) [Instr](#) [Int](#) [IPAddressBox](#) [ItemCount](#) [ItemExtract](#) [ItemInsert](#) [ItemLocate](#) [ItemRemove](#)

[Lcase](#) [Left](#) [LeftOf](#) [Len](#) [ListBox](#) [ListBoxExt](#) [ListLoadedBitmaps](#) [LoadBitmap](#) [Log10](#) [LogE](#) [Logoff](#)

[Max](#) [MessageBeep](#) [MessageBox](#) [Min](#) [MonitorFromPoint](#) [MonitorFromRect](#) [MonitorFromWindow](#) [MonthCalControl](#) [Negate](#) [NumToHex](#) [Pad](#) [PasswordBox](#) [PixelsToDlgUnits](#) [Pow](#) [PrintFile](#) [ProgressBar](#) [PXLcmds](#) [PXLResume](#) [PXLResumeAt](#) [QueryRecycleBin](#)

[Random](#) [RawDataParamBox](#) [RDBCcloseKey](#) [RDBCcreateKey](#) [RDBDeleteKey](#) [RDBEnumKey](#) [RDBOpenKey](#) [RDBQueryKey](#) [RDBQueryValue](#) [RDBSetValue](#) [ReadBitmapRect](#) [ReadRawBitmap](#) [ReadCommPort](#) [ReadTIFFcompressMode](#) [Redraw](#) [RegisterExtLibCmdSet](#) [RegisterUserCommand](#) [RemoveFont](#) [RenameListImage](#) [ReportHistogramStats](#) [ReportMouse](#) [Right](#) [RightOf](#) [RotateRectangle](#) [Run](#) [RunExt](#) [SaveBitmap](#) [SaveRectangle](#) [SaveTIFFcompressMode](#)

[Scrollbar](#) [ScrollbarGetValue](#) [ScrollbarRemove](#) [ScrollbarSetPosition](#) [ScrollbarSetRange](#)

[SendCopyDataMsg](#) [SendKeys](#) [Set](#) [SetMapMode](#) [SetBMWMouse](#) [SetBMWRightMouse](#) [SetColorPalette](#) [SetCommPort](#)

[SetComputerName](#) [SetCtrlMidMouse](#) [SetCtrlMouse](#) [SetCurrentBitmap](#) [SetDbiMidMouse](#) [SetDbiMouse](#)
[SetDbiRightMouse](#) [SetDrawBitmap](#) [SetDrawMode](#) [SetDrawMouse](#) [SetEditControl](#) [SetEnvVariable](#) [SetFontEscapement](#)
[SetKeyboard](#) [SetListBitmapPixel](#) [SetLocalTime](#) [SetMenu](#) [SetMidMouse](#) [SetMouse](#) [SetPopupMenu](#) [SetPriority](#)
[SetRightMouse](#) [SetSendKeysPriority](#) [SetShftMidMouse](#) [SetShftRightMouse](#) [SetSystemTime](#) [SetTextSpacing](#)

[SetVECdrawParams](#) [SetViewportExtent](#) [SetViewportOrigin](#) [SetWaitMode](#) [SetWindow](#) [SetWindowExtent](#) [SetWindowOrigin](#)
[SetWorkingDirBox](#) [ShellAbout](#) [Shutdown](#) [Sin](#) [Sinh](#) [Space](#) [Sqrt](#) [StatusWindow](#) [Str](#) [StrCmp](#) [StrCmpl](#) [StrRepl](#)
[StrReplAll](#) [StrRev](#) [Switch](#)

[Tan](#) [Tanh](#) [TaskBarIcon](#) [TextBox](#) [TextBoxExt](#) [TileBitmapWindows](#) [TimeToASCII](#) [ToolBar](#) [ToolWindow](#)

[Trackbar](#) [TrackbarGetValue](#) [TrackbarRemove](#) [TrackbarSetPosition](#) [TrackbarSetRange](#) [Trim](#) [TrimExt](#)

[TWAIN_AbortAllPendingXfers](#) [TWAIN_AcquireNative](#) [TWAIN_AcquireToClipboard](#) [TWAIN_AcquireToFilename](#)
[TWAIN_CloseSource](#) [TWAIN_CloseSourceManager](#) [TWAIN_CurrentSourceID](#) [TWAIN_DisableSource](#)
[TWAIN_DisableSource](#) [TWAIN_EnableSource](#) [TWAIN_EnableUI](#) [TWAIN_EnumSource](#) [TWAIN_GetBitDepth](#)
[TWAIN_GetBitmapParams](#) [TWAIN_GetCapability](#) [TWAIN_GetCompression](#) [TWAIN_GetCurrentRes](#)
[TWAIN_GetCurrentUnits](#) [TWAIN_GetFrame](#) [TWAIN_GetPixelFormat](#) [TWAIN_GetState](#) [TWAIN_IsAvailable](#)
[TWAIN_LoadSourceManager](#) [TWAIN_OpenDefaultSource](#) [TWAIN_OpenSourceManager](#) [TWAIN_OpenSpecificSource](#)
[TWAIN_PxlVersion](#) [TWAIN_RegisterApp](#) [TWAIN_SelectSource](#) [TWAIN_SetBitDepth](#) [TWAIN_SetCapability](#)
[TWAIN_SetCompression](#) [TWAIN_SetCurrentRes](#) [TWAIN_SetCurrentUnits](#) [TWAIN_SetFrame](#) [TWAIN_SetPixelFormat](#)
[TWAIN_UnloadSourceManager](#)

[Ucase](#) [UpdateHistogram](#) [UpdateProgressBar](#) [UseBackground](#) [UseBrush](#) [UseBrushPattern](#) [UseCaption](#) [UseCoordinates](#)
[UseCursor](#) [UseFont](#) [UsePen](#) [Val](#) [ViewFilterFile](#)

[WaitCommEvent](#) [WaitInput](#) [WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPitch](#) [WAVGetPlayRate](#) [WAVGetVolume](#)
[WAVPlaySound](#) [WAVSetPitch](#) [WAVSetPlayRate](#) [WAVSetVolume](#)

[While Loops](#)

[WinAdjustRect](#) [WinClose](#) [WinExist](#) [WinGetActive](#) [WinGetClientRect](#) [WinGetLocation](#) [WinHelp](#) [WinHTMLHelp](#)
[WinLocate](#) [WinSetActive](#) [WinShow](#) [WinTitle](#) [WinVersion](#)

[WriteBitmapRect](#) [WriteCommPort](#)

[ZoomBitmapWindow](#)

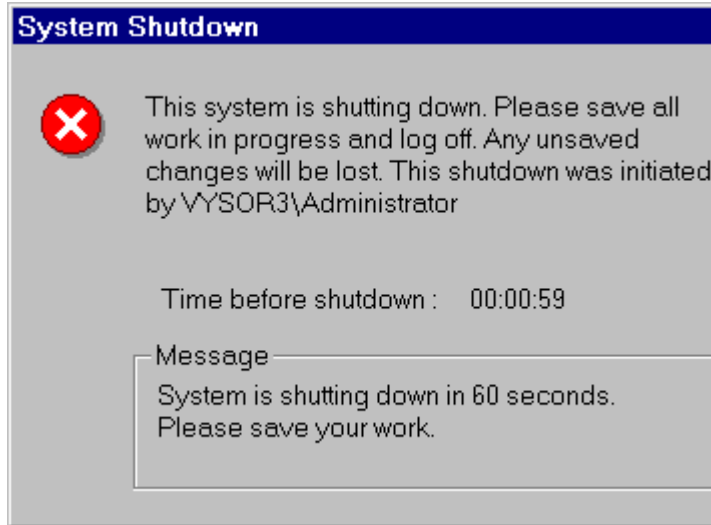
[Image Processing Commands](#)

AbortShutdown

Windows NT command! Cancels the shutdown process initiated by the [Shutdown](#) command.

Syntax: [AbortShutdown](#)

Remarks:



By using PiXCL's [Shutdown](#) command, you can have PiXCL display a message box like the one at left, indicating that the system is about to shut down in a specified number of seconds. You can cancel the shutdown process and remove the dialog box by using the [AbortShutdown](#) command. See the [Shutdown](#) command for an example.

If this command is used in Windows 95, a message box appears to inform you that the command is not supported. The program will then continue.

Related Commands:

[Logoff](#) [ExitWindows](#) [Shutdown](#)

AboutPiXCL

The standard About box can be made to appear by clicking on the [&Info](#) menu item, unless it has been disabled by the [InfoMenu\(REMOVE/ENABLE\)](#) command. The [AboutPiXCL](#) command can be used if this standard about box is needed at times when the &Info menu item has been disabled, or the entire menu bar has been deleted with the [SetMenu\(\)](#) command.

Syntax: [AboutPiXCL](#)

Parameters: None

Related Commands:

[InfoMenu](#) [SetMenu](#)

AboutUser

A user defined About box can be made to appear. The title bar string and two user defined strings can be defined if required.

Syntax: `AboutUser(Title$,AboutBox_1$,AboutBox_2$)`

Parameters:

- Title\$** Title string that appears in the About box. This can be set to a null string i.e. `Title$ = ""`, and no title sting will be displayed. The maximum string length is 128 characters.
- AboutBox_1\$** Up to two lines of text. This would generally be the name of the new application. If this string is set to null (i.e. ""), the region in the About box will appear blank.
- AboutBox_2\$** Up to four lines of text. This would generally be details of the new application, such as technical support contact information, or more data about the program. If this string is set to null (i.e. ""), the region in the About box will appear blank.

Example

```
AboutUser("Structured Program Loops with PiXCL 4.0",  
  "FOR - NEXT and WHILE - ENDWHILE structures in PiXCL",  
  "Sample PiXCL 4.0 program showing variety of built in icons  
  drawn in the client area, depending on the loop parameters")
```

Related Commands:

[ShellAbout](#)

Acos

Floating Point math library function. Calculate the arc cosine of an angle in radians.

Syntax: *Acos*(*Angle&*, *Value&*)

Parameters:

Angle& The angle in radians
Value& The result of the function.

Related Commands:

[Asin](#) [Atan](#)

AddFont

The [AddFont](#) command adds the font resource from the specified file to the Windows font table. The font can subsequently be used for text output by any Windows-based application.

Syntax: [AddFont\(FontFilename\\$,Result\)](#)

Parameters:

[FontFilename\\$](#) A valid font file filename. The filename may specify either a .FON font resource file, a .FNT raw bitmap font file, a .TTF raw TrueType file, or a .FOT TrueType resource file.

[Result](#) 1 if the operation succeeded, otherwise 0. If the font is already installed, [Result](#) also returns 0.

Related Commands:

[DrawText](#) [DrawNumber](#) [RemoveFont](#) [UseFont](#)

Ansi

Returns the ANSI code for the first character in a string.

Syntax: *Ansi(String\$,Code)*

Parameters:

String\$ A string consisting of one or more characters.

Code An integer variable that will contain the ANSI character code for the first character in *String\$*.

Remarks:

Although Windows NT and 95 support both the ANSI and Unicode character sets, this version of PiXCL supports only the ANSI set.

There are 256 characters in the ANSI set, numbered 0 to 255. Characters on your keyboard are represented by numeric values between 32 and 126--for example, the ANSI code for **T** is 84. ANSI characters outside this range represent special characters, such as fractions and accented characters. You can use the Windows CHARMAP accessory to view all the available characters and extended characters.

Example:

This example draws the number 65--the ANSI code for the letter *A*--at the point (10,10) in the PiXCL window.

```
Ansi("Alphonse",Result)
DrawNumber(10,10,Result)
WaitInput()
```

Related Command:

[Chr](#)

AppWindowHandle

PiXCL can be used to call other EXE programs with command line arguments, which may need to send messages back to the PiXCL application. It also reports the interpreter type (**PiXCL** or **geoPiXCL**) and the version numbers.

Syntax: *AppWindowHandle(Handle,Handle\$, geoPiXCLFlag,Major,Minor)*

Parameters:

Handle The window handle of the current PiXCL application.

Handle\$ *Handle* expressed as a string variable.

geoPiXCLFlag Returns 1 if the application is using the more extensive **geoPiXCL** interpreter, otherwise it returns 0 for all versions of PiXCL.

Major, Minor The major and minor version numbers of the interpreter.

Remarks:

This command will primarily be of use to programmers writing other applications in Visual Basic, C or C++. If *Handle\$* is passed to another application, you can use the value in Windows SendMessage commands. For example, you can send a WM_COPYDATA message to a PiXCL application that tells it to start processing at a specific label. This is useful when you want an extension function to send a progress message, and have the PiXCL application display a progress bar.

Related Commands:

None.

Array

PIXCL 5 Command. In PIXCL 5.0 and later, 32-bit integer, 32-bit floating point and string arrays are supported. PIXCL 5.1 added 64-bit integer and 64-bit double arrays. Unlike integer, floating point and string variables that can be set and reset anywhere in a program (e.g. strings and the `FreeVar` command), arrays have to be created or deleted explicitly because of the memory allocation and deallocation requirements.

Syntax: `Array(VariableName[Size],Result)` for integer arrays
`Array(VariableName$[Size],Result)` for string arrays
`Array(VariableName&[Size],Result)` for floating point arrays
`Array(VariableName#[Size],Result)` for 64-bit integer arrays
`Array(VariableName#[Size],Result)` for 64-bit double arrays

Parameters:

VariableName The name of the array. This must not be the same as any other variable of the same type.

Size The number of elements in the array, and must be at least 1. The current maximum size of an array is arbitrarily set to 2048K elements. Note that string array elements are actually pointers to the first character in a string of arbitrary length.

Result If the creation of the array was successful, **Result** returns the same value as **Size**, otherwise 0. The usual cause of a 0 return is that **Size** is greater than the maximum size, or is zero or a negative number.

Remarks:

Integer array elements are initialized to 0.
String array elements are initialized to "".
Float array elements are initialized to 0.0.

If an array variable has been created with the `Array` command, and you issue another command with the same variable name and arbitrary size, the existing array is DELETED and the new array created with zeroed entries. The first array can also be deleted with a `FreeArrayVar` or `FreeArrayVarAll` command and then the new array can be created.

When a PIXCL application terminates, all array variables and memory get automatically deleted. From a coding point of view, it's a good idea to have a `FreeArrayVarAll` command to ensure that cleanup occurs.

Using array variables:

Array variables are referenced with a 0 based index enclosed in square brackets e.g. `VariableName[index]` or `VariableName [index]`. Note the space after the `VariableName` and `[index]` : both are supported. We suggest that you use the `VariableName[index]` for code clarity.

Array variable elements can be used anywhere that the same type of variable is used. For example, let's say you have the same image file name in both `ImageName1$` and `ImageNameArray$[1]`

```
DrawBitmap(x,y,ImageName1$ ) and  
DrawBitmap(x,y, ImageNameArray$[1] )
```

will have the same effect.

If you want to implement multi-dimensional arrays, create a set of name related arrays e.g.

```
Array(Vector_X [Size],Size)  
Array(Vector_Y [Size],Size)  
Array(Vector_Z [Size],Size)
```

Common Programming Errors

You will get a Syntax Error if you

- try to access an array variable without first declaring it.
- try to access an array element with an out of range index or negative index.

- forget to add a \$ or & to a string or float arrayname when accessing an array element.

Related Commands:

[FreeArrayVar](#) [FreeArrayVarAll](#)

Asin

Floating Point math library function. Calculate the arc sine of an angle in radians.

Syntax: *Asin*(*Angle&*, *Value&*)

Parameters:

Angle& The angle in radians
Value& The result of the function.

Related Commands:

[Acos](#) [Atan](#)

Atan

Floating Point math library function. Calculate the arc tangent of an angle in radians.

Syntax: *Atan*(*Angle&*, *Value&*)

Parameters:

Angle& The angle in radians
Value& The result of the function.

Related Commands:

[Acos](#) [Asin](#)

Average

Floating Point math library function. Calculate the average of a list of real numbers.

Syntax: *Average(Number_1&,...,Number_n&, Value&)*

Parameters:

Number& Each of the list of numbers. These numbers can be integers (not integer variables).
Value& The average of the list.

Related Commands:

[Max](#) [Min](#)

AutomateApp

PIXCL 5 command: Many programs, notably Microsoft Office applications like Word and Excel, and many other third party applications, are what is called Automation servers. That is, it is possible for a client application (here, PiXCL and geoPiXCL, but also Visual Basic, C++ and others) to programmatically control the server, using what are called COM interfaces. You will need to know the exact Automation command(s) syntax, which is supplied by the server developer.

Syntax: *AutomateApp(Interface, Command\$, CommandMODE, Integer|Float&|String\$, ArgumentType, ..., ..., ..., ..., Result\$ | Result | Result&)*

Parameters:

Interface A COM interface initially acquired from a call to [CreateCOMInstance](#), but possibly also from a call to this function.

Command\$ The Automation command string.

CommandMODE One of the following:

METHOD A function that invokes some process in the target. It may or may not have arguments.

PROPERTYPUT A function that sets a property on the target. Usually takes one argument.

PROPERTYGET A function that gets a property from the target, and takes one argument.

The Automation command arguments come in pairs. Any number of argument pairs can be specified.

Integer|Float&|String\$ A static value or variable which is a representation of the argument value. For boolean values in strings, use “@TRUE” or “@FALSE”. Using @TRUE or @FALSE as an integer value here will result in a syntax error: use 1 or 0 instead.

ArgumentType The type of the argument required by the Automation command. The representation is converted internally by PiXCL. Currently supported types are:

VT_NULL where a null value is required.

VT_BSTR where a BSTRING is required.

VT_I2, VT_I4 2 byte and 4 byte integers respectively.

VT_R4 where a 4 byte float variable is required.

VT_R8 where an 8 byte double variable is required.

Result\$|Result|Result& The return variable of the type required by the Automation command that was specified. Since PiXCL and geoPiXCL support three variable types only, choose the most appropriate type for *Result*. E.g.

BOOLs, Handles, Pointers, Interfaces == integer

Floats or doubles == float&

Strings == string\$

The return value is also used as a negative integer error return, represented in the variable type. Hence an error return of -1 can also be “-1” and -1.0.

-1 = UNKNOWNINTERFACE: the COM interface specified is unknown.

-3 = MEMBERNOTFOUND: a method requested is unrecognized.

-4 = PARAMNOTFOUND: a property requested is unrecognized.

-5 = TYPEMISMATCH: a type specified for an argument is incorrect.

-6 = UNKNOWNNAME: the server does not recognize a specified name.

Remarks:

A common programming error is to add extra argument pairs to the command. Please consult your target application Automation command syntax. Another common error is setting the wrong argument name or type. This will generally result in PiXCL halting with a syntax error, which of course, it is.

Related Commands:

[CreateCOMInstance](#) [ReleaseCOM](#)

AutoProgressBar

When PiXCL loads a bitmap into the image list the first time, or performs some of the image processing options, a progress bar is displayed by default along the bottom of the client area as the operation progresses. This progress bar display can be disabled if desired, with the [AutoProgressBar](#) command.

Syntax: [AutoProgressBar\(ENABLE | DISABLE\)](#)

Parameters:

[ENABLE](#) The default state.
[DISABLE](#) A progress bar is not displayed.

Remarks:

This command can be issued at any time during a program's operation. It does not affect the [ProgressBar](#) command.

Related Commands:

[DrawBitmap](#) [DrawSizedBitmap](#) [DrawTrBitmap](#) [DrawTrSizedBitmap](#) [DrawZoomedBitmap](#)

Beep

Sounds the bell. If a sound card is installed, the current sound for the bell will be played.

Syntax: [Beep](#)

Example:

This program puts up a message box with Yes, No, and Cancel buttons. It then beeps once if you select Yes, twice for No, and three times for Cancel.

```
{Put up a message box}
  MessageBox (YESNOCANCEL, 1, QUESTION,
             "Did you vote?", "Question", Button)
{Test for which button was selected}
  If Button=1 Then Goto Beep1
  If Button=2 Then Goto Beep2
  {Else} Beep
  WaitInput(500)    {Pause for 1/2 second}
Beep2:
  Beep
  WaitInput(500)
Beep1:
  Beep
```

Related Command:

[MessageBeep](#) [WAVPlaySound](#)

BitFlag

PIXCL 5 command. Individual bits in an integer value are sometimes used as flags, for example returns from other hardware via the serial port. The [BitFlag](#) command provides the means to check if any bit is set.

Syntax: [BitFlag\(Number,Bit,BitValue\)](#)

Parameters:

| | |
|--------------------------|---|
| Number | The number to be checked. |
| Bit | A number in the range 0-31. Out of range or negative numbers are converted to positive and mod32 taken. |
| BitValue | 0 or 1 |

Related Command:

[Checksum](#)

BitmapToTxt

PIXCL 5 command. It is sometimes helpful to be able to write the pixel values of an image out to a text file. [BitmapToTxt](#) accepts 8 bits per pixel only. The output is TAB delimited between values, with a CR-LF after each image line. Use this command with care, as it can create very large files up to 4 times larger than the original image.

Syntax: [BitmapToTxt\(Imagename\\$,Filename\\$,Result\)](#)

Parameters:

| | |
|-----------------------------|--|
| Imagename\$ | The name of an image loaded into the PIXCL image list e.g. with the LoadBitmap command. Must be 8 or 24 bits per pixel, or no file is created. |
| Filename\$ | The name of the text file to be written. An existing file is overwritten. A summary of the image parameters is written to the first line of the file. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

BMWRedraw

Use this command to refresh the contents of any visible bitmap windows

Syntax: [BMWRedraw](#)

Parameters:

None

Related Commands:

[BMWSysCmdEndAt](#) [BMWSetTitle](#) [ChangeBmwImage](#)

BMWSysCmdEndAt

A bitmap window can be closed by clicking the Close button, which also updates the internal PiXCL records. You can optionally have some processing done by specifying an individual label for each window. You can change the defined label at any time. Setting the label to NULL i.e. disabling the Close button processing requires the bitmap window to be deleted and redrawn.

Syntax: [BMWSysCmdEndAt\(WindowID,Label\)](#)

Parameter:

[WindowID](#) A valid window ID from [DrawBitmapWindow](#).

[Label](#) The jump-to label name. If [Label](#) does not exist in the script, a syntax error is generated.

Related Commands:

[ChangeBmwImage](#) [PXLResumeAt](#) [SysCmdEndAt](#) [WinResizeAt](#)

BMWinTitle

A bitmap window created by the [DrawBitmapWindow](#) command places the selected image filename in the title bar by default. When you need to change or add to the title, such as adding more information like the current zoom factor, or last selected mouse position, the [BMWinTitle](#) command should be used.

Syntax: [BMWinTitle\(WindowID,NewTitle\\$\)](#)

Parameters:

[WindowID](#) The ID number returned by the [DrawBitmapWindow](#) command. If the ID does not refer to an existing window, the command is ignored.

[NewTitle\\$](#) The new text that should appear in the title bar.

Remarks:

The bitmap window should have been created with the [CAPTION_SIZE](#) or [CAPTION_NOSIZE](#) tokens, as these make the title bar visible.

Related Commands:

[ChangeBmwImage](#) [DrawBitmapWindow](#) [SetBMWMouse](#) [ZoomBitmapWindow](#)

BMWUseCursor

PIXCL5 command. Changes the mouse cursor in all displayed bitmap windows to one of the predefined Windows or PiXCL cursors. This is independent of the cursor set by the [UseCursor](#) command that affects the displayed cursor within the main PiXCL client area.

Syntax:

[BMWUseCursor](#)(APPSTARTING / ARROW / CROSS / IBEAM /
ICON / NO / SIZE / SIZEALL / SIZENESW /
SIZENS / SIZENWSE / SIZEWE / UPARROW /
WAIT / ZOOM / CROSSHAIR)

Parameters:

| | |
|-----------------------------|--|
| APPSTARTING | Standard arrow and small hourglass. |
| ARROW | Standard arrow. |
| CROSS | Crosshair. |
| IBEAM | Text I-beam. |
| ICON | Empty icon. |
| NO | Slashed circle. |
| SIZE | Four-pointed arrow. |
| SIZEALL | Same as SIZE. |
| SIZENESW | Double-pointed arrow pointing northeast and southwest. |
| SIZENS | Double-pointed arrow pointing north and south. |
| SIZENWSE | Double-pointed arrow pointing northwest and southeast. |
| SIZEWE | Double-pointed arrow pointing west and east. |
| UPARROW | Vertical arrow. |
| WAIT | Hourglass. |
| ZOOM | Magnifying glass. |
| CROSSHAIR | Crosshair for selecting a pixel. |

Remarks:

The [BMWUseCursor](#) command has an effect only under these circumstances:

- The new cursor is different from the previous cursor.
- PiXCL is waiting for mouse input; a [SetBMWMouse](#), [SetBMWMidMouse](#) or [SetBMWRightMouse](#) is in effect.

Related Commands:

[ChangeBmwImage](#)

[SetCtrlMouse](#) [SetMouse](#) [SetDblMouse](#) [SetRightMouse](#) [WaitInput](#)

Button

Lets you create standard gray rectangular 3-D pushbuttons, round radio buttons, check boxes and group boxes in any combination and place them anywhere within a PiXCL window. When the program is pausing for input and you click on a button, the program branches to the label associated with that button.

Syntax:

`Button()` to clear any existing buttons.

or

```
Button(Btn1_x1,Btn1_y1,Btn1_x2,Btn1_y2, STYLE,ButtonText1$,Label,  
Btn2_x1,Btn2_y1,Btn2_x2,Btn2_y2, STYLE,ButtonText2$,Label,  
.  
Btnn_x1,Btnn_y1,Btnn_x2,Btnn_y2, STYLE,ButtonTextn$,Label)
```

Parameters:

| | |
|-------------------------------|--|
| <code>Btnx_x1, Btnx_y1</code> | The upper-left corner of a button. |
| <code>Btnx_x2, Btnx_y2</code> | The lower-right corner of a button. |
| STYLE token: | Selects the type of button: |
| PUSH | standard pushbutton with text label. |
| PUSHBMP | PIXCL 5 . Pushbutton with custom bitmap file (BMP or RLE). |
| PUSHICO | PIXCL 5 . Pushbutton with ICO icon file or PiXCL or SHELL32 icon. |
| RADIO | radio button. |
| AUTORADIO | toggleing radio button. |
| CHECK | check box. |
| AUTOCHECK | toggleing check box. |
| GROUP | group box. |
| NULL | PIXCL 5 . Draws a flat style button. See the Label to a dummy e.g. <code>Wait_for_Input</code> . |
| <code>Text\$</code> | The text to be displayed within the button control. For the PUSHBMP mode, this becomes the name of the BMP or RLE disk file that implements the button. For PUSHICO mode, use an ICO filename, or use the same ICON identifiers as used with the DrawIcon and DrawShellIcon commands, preceded by a # character e.g. “#ICON07” or “#SCANNER” or “#SHICON33”. |
| <code>Label</code> | The label you want to branch to when the user clicks on the button. |

Remarks:

When PiXCL encounters a **Button** command in your program, it immediately draws the specified button styles on the screen. You can have multiple button styles at any time, but only one Button command is valid at any instant.

Push buttons, radio buttons and checkboxes are mouse active, and will branch to the specified label. A groupbox is not mouse active, and the label specified will not be accessed. In addition, group boxes are not windows like the check/radio/push buttons, but are drawn into the client area. This means that the [SetROPcode](#) command can have an effect on the appearance.

For example, the default system background color is light gray (192,192,192), and this is the color of pushbuttons and check/radio button text. With the ROP code set to the default SRCCOPY, a group box will show the existing background in the box. If the ROP code is DSTCOPY, the group box will be filled with the current system background color.

The `Text$` string appears in various ways depending on the button control style. In a standard Pushbutton, it is the title that appears on the button. In Radio buttons and Checkboxes, it is the text that appears next to the control. You must adjust the x,y coordinates of the control to include the size of the text, since the control is actually a separate small window. In a groupbox, `Text$` is the title that appears in the group border.

`Text$` is also affected by the current font style and size, but will always be displayed in black, regardless of the color specified in the [UseFont](#) command.

Although the buttons are visible, you cannot actually access them until the program is pausing for input. [To have a program pause for input, you must use the `WaitInput()` command.] At that time, if you click on a button, control transfers to the `Label` associated with the button.

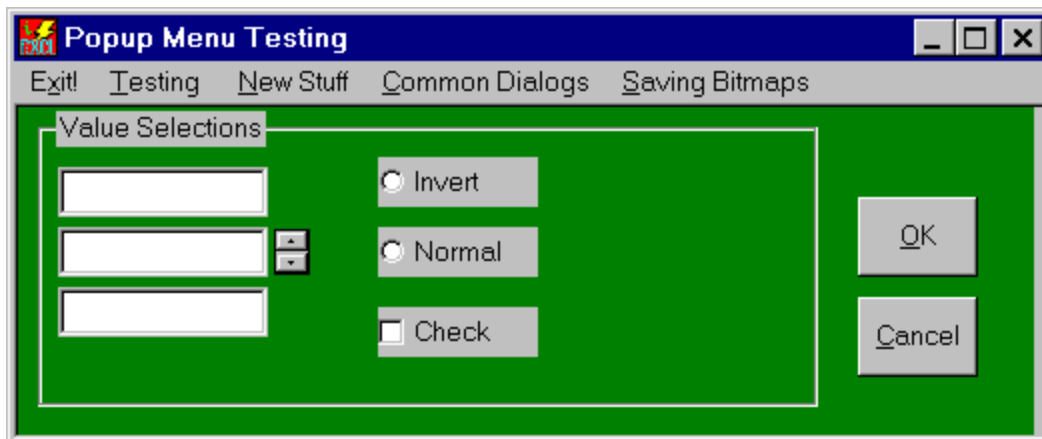
- A Button command remains in effect until any of the following occurs:
- You use another Button command.
- You use Button without any parameters (i.e. `Button()`) to remove all existing buttons from the screen.
- The program ends, and all button resources are freed, and the memory returned to Windows.

When creating a `Text$` argument, you can place an `&` in front of a letter to have the letter appear underlined. For example, the parameter `"&Cancel"` underlines the letter `C` in the Cancel button.

PIXCL automatically provides mouse support for buttons created with the Button command. If you want the user to be able to select a button using the keyboard, you'll need to use the `SetKeyboard` command (see the example).

Example:

The following program fragment places three buttons on the screen, as shown in the figure below. Observe how the radio box, checkbox and groupbox strings are drawn in the current sytem background color.



Creating 3-D buttons, Radio Buttons, Checkbox and GroupBox with the Button command. Edit controls are also shown.

New_Buttons:

```
UseBackground(TRANSPARENT,0,192,192)
UseFont("MS Sans Serif",0,19,NOBOLD,NOITALIC,NOUNDERLINE,255,255,0)
SetEditControl(20,30,125,55,STRING,0,0,Edit1$,
               20,60,125,85,NUMBERUD,100,0,Edit2$,
               20,90,125,115,PASSWORD,0,0,Edit3$)
```

```
Button(420,45, 480, 85,PUSH,"&OK",New_Buttons_End,
       420,95, 480, 135,PUSH,"&Cancel",New_Buttons_End,
       180,25,260, 50,AUTORADIO,"Invert",New_Buttons_Wait,
       180,60,260, 85,AUTORADIO,"Normal",New_Buttons_Wait,
       180,100,260,125,AUTOCHECK,"Check",New_Buttons_Wait,
       10,2,400, 150,GROUP,"Value Selections",New_Buttons_End)
```

New_Buttons_Wait:

```
WaitInput()
```

New_Buttons_End:

```
SetEditControl()  
Button()  
Goto Wait_for_Input
```

Related Commands:

[SetKeyboard](#), [WaitInput](#), [UseFont](#), [SetEditControl](#)

See also [Making 3D BitMapped buttons](#), [ToolBars and ToolWindows](#)

Breakpoints

PIXCL 5 command. If the PXLdebug.dll is installed, you can enable a dialog that presents all the integer, float, string, array and bitmap records when the associated SetBreakpoint command is encountered.

Syntax: `Breakpoints(@FALSE | @TRUE)`

Parameter:

`@TRUE` Enable breakpoints and load PXLdebug.dll. If the DLL is already loaded, it does not get loaded again.

`@FALSE` Disable breakpoints and unload PXLdebug.dll.

Related Command:

[SetBreakpoint](#)

ChangeMenuItem

This function lets you check, uncheck, gray, or enable an existing PiXCL menu item. Menu items are often grayed when the selection is not currently meaningful, such as when another function must be executed first. For example, menu items that perform image processing functions should be grayed until an image has been loaded into the PiXCL image list.

Syntax:

`ChangeMenuItem(Item,$,CHECK/UNCHECK/GRAY/ENABLE,Result)`

Parameters:

| | |
|----------------|--|
| <i>Item</i> \$ | The pop-up menu item to be modified. The text of the menu item must be spelled exactly as it was when declared with the SetMenu command, including any leading or trailing spaces. If the menu item has an underlined letter, be sure to place an & in front of the letter when specifying <i>Item</i> \$. |
| CHECK | Places a checkmark next to the menu item. |
| UNCHECK | Removes a checkmark from a menu item. |
| GRAY | Disables (grays) a menu item. |
| ENABLE | Enables a grayed menu item. |
| <i>Result</i> | An integer variable that indicates the outcome of the change. If the change was successful, this variable is assigned a value of 1. If the menu item was not found, this variable is assigned a value of 0. If the menu has been deleted with a SetMenu() command, ChangeMenuItem returns 0. |

Remarks:

You cannot change the appearance of a top-level menu item with this function, only pop-up menu items.

If more than one menu item shares the same text, PiXCL will apply the attribute to the last such item in the menu.

If you have similar pop-up menu item names, you can differentiate them by using trailing spaces. You can also differentiate otherwise identical text with the "&" character to underline a different accelerator key.

This command also works with popup menus created with the [SetPopupMenu](#) command.

Example:

The following code fragment creates a simple menu with one top-level menu item--Yearend--and beneath it a pop-up menu with two menu items--Tax and Book. The [ChangeMenuItem](#) command is then used to place a checkmark next to Tax menu item.

```
{Set up the menu}
  WaitInput(100) { needed only the first
    time the SetMenu command is used}
  SetMenu("&Yearend", IGNORE,
    "&Tax", Tax,
    "&Book", Book,
    ENDPOPOP)

{Check the Tax menu item}
  ChangeMenuItem("&Tax", CHECK, Result)

Wait_for_input:
  WaitInput()
```

Tax:

{More code goes here...}

Book:

{More code goes here...}

Related Commands:

[SetMenu](#) [GetMenuStatus](#) [SetPopupMenu](#) [ToolBar](#) [ToolWindow](#) [ChangeToolBarBtn](#) [GetToolBarBtnStatus](#)

ChangeToolBarBtn

This function lets you check, uncheck, press, disable or enable an existing PiXCL toolbar or toolwindow button, in the similar way to the [ChangeMenuItem](#) command.

Syntax:

[ChangeToolBarBtn](#)(*ToolBar\$,ButtonIndex,DISABLED/ENABLED/CHECKED/PRESSED,Result*)

Parameters:

Name\$ If NULL (""), refers to the current [ToolBar](#), if present. Otherwise, *Name\$* refers to a current [ToolWindow](#) title. If the toolbar or toolwindow is not present, the command is ignored. *Result* returns 0.

BtnIndex Button index, starting from 1, as defined in the [ToolBar](#) command in effect. [SEPARATOR](#) regions are counted as buttons.

state_TOKEN [ENABLED](#) | [CHECKED](#) | [PRESSED](#) | [DISABLED](#)

Result 1 if the button is in the defined [state](#), otherwise 0.

Remarks:

If you send a change state message to a [SEPARATOR](#) region, it is ignored and *Result* returns 1, because it will successfully do nothing.

Example:

See the sample program `toolbars.pxl`.

Related Commands:

[GetToolBarBtnStatus](#) , [ToolBar](#) , [ToolWindow](#)

ChangeBMWimage

PIXCL 5 command. PIXCL supports up to eight bitmap windows with independent zoom and roam, from images loaded into the PIXCL image list. It is possible to replace the image displayed in a bitmap window without closing that window and reloading it.

Syntax: [ChangeBMWimage\(WindowID, Filename\\$, Result\)](#)

Parameters:

| | |
|----------------------------|---|
| WindowID | The bitmap window ID from a DrawBitmapWindow command. |
| Filename\$ | The image that is to be displayed in the window. If the image is not already loaded into the PIXCL image list, it is automatically loaded. If the the image cannot be found, the command has no effect. |
| Result | 1 if the operation was successful, otherwise 0. |

Remarks:

All this command does is replace the name of the image that is to be displayed in the bitmap window, then redraw the window. The scroll bars may change if the new image dimensions are different, but no changes are made to the zoom factor.

Related Commands:

[BMWinTitle](#) [CloseBitmapWindow](#) [DrawBitmapWindow](#) [FlashBMWindow](#)

CheckBitmapFormat

When you need to attempt to load a image file from the disk, it can be helpful to check if the format is one of those supported by the library. DrawBitmap and LoadBitmap do not make this check, but LoadBitmapExt does.

Syntax: `CheckBitmapFormat(Filename,$Supported)`

Parameters:

`Filename`\$ The disk image file.

`Supported` If the file format is recognized, returns 1, or 0 if the format is unknown or the file is not found.

Related Commands:

[DrawBitmap](#) [LoadBitmapExt](#)

Checksum

PIXCL 5 command. Generate the checksum of a string.

Syntax: `Checksum(String$,ChecksumValue)`

Parameters:

String\$ The string to be used to calculate the checksum. The ASCII character values are summed.

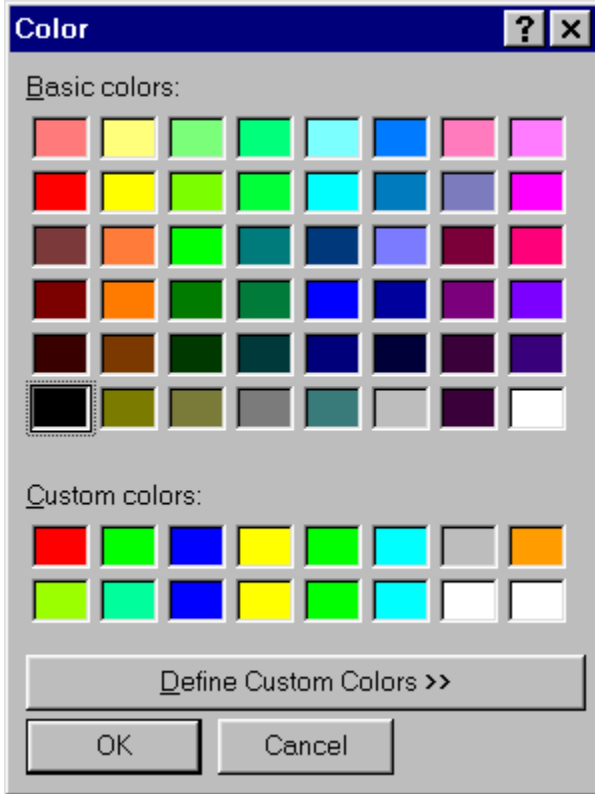
ChecksumValue The lower 4 bits of the sum are returned as an integer.

Related Command:

[BitFlag](#)

ChooseColor

The standard COMDLG32 library call can be used to display the Color select dialog box in various formats. The chosen color is returned as an RGB triplet, and can be used in any PIXCL command that uses color, such as [DrawFlood](#), [UsePen](#), [UseBrush](#), [UseFont](#) or [UseBackground](#).



The standard ChooseColor dialog.

The [ChooseColor](#) commands are handy when you want to offer the user a defined choice of color, for example to set the color of a pen, brush, or any situation where a color value is required.

You can also create a set of user defined colors with the [CustomColor](#) command. This set is static for the duration of the PIXCL application, that is, it remains as you set or reset it each time you exit from the [ChooseColor](#) dialog. For PIXCL 4.22 and later, the [ChooseColor](#) command has been extended with the ability to modify the dialog window position, title, basic and custom strings, as shown in the dialog shown above.

Syntax: [ChooseColor](#)(STYLE_TOKEN,*Red,Green,Blue*)
[ChooseColor](#)(STYLE_TOKEN,*Red,Green,Blue,X,Y,Title\$,Basic\$,Custom\$*)

Parameters:

STYLE_TOKEN

- STD display a small color select dialog with User Defined Color button enabled.
- SMALL displays a small color select dialog with User Defined Color button disabled.
- SMALLRGB displays a small color select dialog with default RGB color selected.
- FULL displays a large color select dialog.
- FULLRGB displays a large color select dialog with default RGB color selected

Red, Green, Blue *Red, Green* and *Blue* set the default color, and return the selection values in the range 0-255.

X,Y The top-left corner position for the dialog, in client area coordinates. These coordinates can be negative.

Title\$ A user defined title for the dialog. If this is set to null, the default “**Color**” string is used.

Basic\$ A string that replaces the default “**Basic Colors:**”. If this is set to null, the default string is used.

Custom\$ A string that replaces the default “**Custom Colors:**”. If this is set to null, the default string is

used.

Remarks:

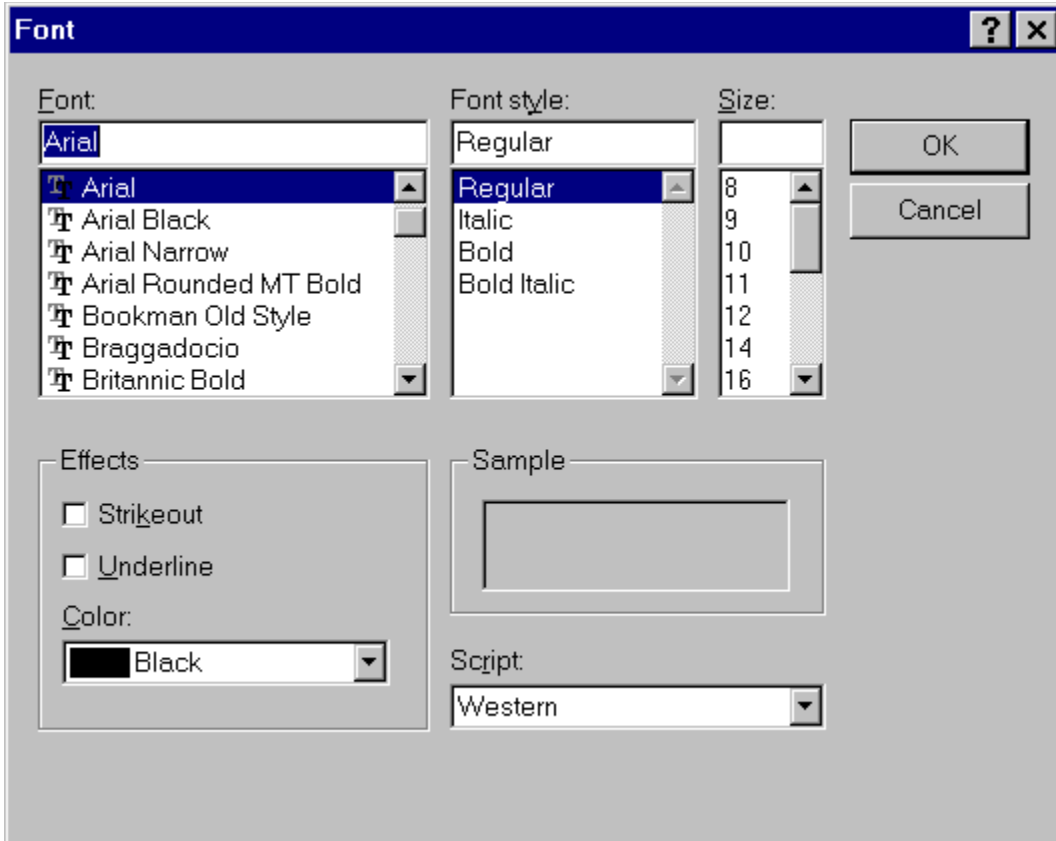
If you select the **Cancel** button, *Red* returns a -1 value.

Related Commands:

[CustomColor](#)

ChooseFont

The standard COMDLG32 library call for choosing a font can be accessed to dynamically select the font style for text output. The command parameters are both inputs and outputs, and can be used to initialize the dialog controls.



Syntax: [ChooseFont\(Font\\$, Width, Height, r, g, b, Bold, Italic, Underline, Strikeout\)](#)

Parameters:

[Font\\$](#)

The name of the selected font or currently available font. If you set this variable before calling [ChooseFont](#), the name appears in the Font edit control.

[Width, Height](#)

The default input and returned selected width and height. You can set these to 0 as defaults. If you specify non-zero values, the equivalent point size is calculated and displayed in the Size control.

[r, g, b](#)

The default input and returned color for the selected font.

[Bold, Italic,](#)

[Underline,](#)

[Strikeout](#)

These are also input and output values. If the values are zero, the style is not asserted. If asserted, the value is set to 1.

Remarks:

If you press the **Cancel** button, [Font\\$](#) returns a null string.

Related Commands

[AddFont](#) [UseFontExt](#) [DrawText](#) [DrawTextExt](#) [RemoveFont](#)

Chr

Returns a one character string based on the specified ANSI code.

Syntax: *Chr(Code,String\$)*

Parameters:

Code An integer from 0 to 255.

String\$ A string variable that will contain the ANSI character.

Remark:

If *Code* is outside the range 0 to 255, PiXCL will issue an error message.

Example:

This example draws the letter Z -- the ANSI character assigned to code number 90 -- at the point (10,10) in the PiXCL application client area.

```
Code=90
Chr (Code, Char$)
DrawText (10,10,Char$)
WaitInput ()
```

Related Command:

[Ansi](#)

ClearBitmapOptions

PiXCL 5 command. JPG, PNG and TIF images that have text fields defined in the file headers load this data into Options fields. You can remove these Options with this command. Renaming the image with [RenameListImage](#) has the same effect.

Syntax: [ClearBitmapOptions\(ListImage\\$, Result\)](#)

Parameters:

[ListImage\\$](#) An image loaded into the PiXCL image list.
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[GetJPGOptions](#) [GetPNGOptions](#) [GetTIFFOptions](#) [RenameListImage](#) [SetJPGOptions](#) [SetPNGOptions](#) [SetTIFFOptions](#)

ClearCommPort

This command clears the selected communications port, deletes any outstanding data in the buffers, and resets the port error indicator.

Syntax: [ClearCommPort\(COMx\)](#)

Parameters:

COMx Port, where x = 1 – 4 for standard PCs. **PIXCL 5:** If you have a multiple COM port board installed, ports 5-13 are supported.

Remarks:

The same effect can be achieved by sending SetCommPort.

Related Commands:

[EscCommFunction](#) [GetCommPort](#) [ReadCommPort](#) [SetCommPort](#) [WaitCommEvent](#) [WriteCommPort](#)

ClipboardAppend

Copies text to the end of the Windows Clipboard's existing text.

Syntax: `ClipboardAppend(String$,Result)`

Parameters:

String\$ The text you want to append to the Clipboard's existing text.

Result An integer variable that indicates the outcome of the operation. If the operation was successful, this variable is assigned a value of 1. If the operation fails either because of a shortage of memory or the presence of a non-text Clipboard format, this variable is assigned a value of 0.

Remark:

The `ClipboardAppend` command will set **Result** to 0 if the Clipboard doesn't already contain text.

Example:

This example reads text from the Clipboard and displays it on the screen for 1 second. It then creates a second copy of the text in Clipboard using the `ClipboardAppend` function.

```
{Get the Clipboard's contents and display them}
ClipboardGet(Contents$,Result)
If Result = 0 Then Beep | End
Gosub Drawit
WaitInput(1000)

{Create a second copy of the Clipboard's contents}
ClipboardAppend(Contents$,Result)
If Result = 1 Then Goto Getit
Beep | End

{Draw the Clipboard's contents again}
Getit:
ClipboardGet(Contents$,Result)
If Result = 0 Then Beep | End
Gosub Drawit
WaitInput()

{Drawing subroutine}
Drawit:
Len(Contents$,Length)
DrawText(10,10,"Length of Contents$ is")
DrawNumber(100,10,Length)
DrawText(10,20,"Contents$ contains")
DrawText(100,20,Contents$)
Return
```

Related Commands:

[ClipboardEmpty](#) [ClipboardGet](#) [ClipboardPut](#)

ClipboardEmpty

Empties the Windows Clipboard of its contents. This includes any bitmap, text or other data.

Syntax: [ClipboardEmpty](#)

Comment:

You don't need to empty the Clipboard before using [ClipboardPut](#). This command will empty the Clipboard automatically before copying text to it.

Related Commands:

[ClipBoardAppend](#) [ClipboardGet](#) [ClipboardPut](#)

ClipboardGet

Reads text from the Windows Clipboard.

Syntax: `ClipboardGet(String$,Result)`

Parameters:

String\$ A string variable that will receive the Clipboard's text.

Result An integer variable that indicates the outcome of the operation. If the operation was successful, this variable is assigned a value of 1. If the operation failed, this variable is assigned a value of 0.

Remarks:

The `ClipboardGet` command will fail to read the Clipboard's contents under the following circumstances:

- The Clipboard contains no text.
- If there is not enough memory to hold the text. (To free string variables from memory, use the `FreeVar` or `FreeVarAll` command.)

In either case, `String$` returns empty.

Example:

This example reads text from the Clipboard (if there is any) and displays it in a message box.

```
ClipboardGet (Contents$,Result)
If Result=0 Then Contents$ = "Clipboard has no text"
MessageBox (OK,1,INFORMATION,
    Contents$,"Clipboard text",Button)
```

Related Commands:

[ClipboardAppend](#) [ClipboardEmpty](#) [ClipboardPut](#)

ClipboardGetBitmap

This command takes an image that is loaded into the Clipboard, and pastes it the PiXCL image list. All images loaded into PiXCL are Windows bitmaps, regardless of the type of image that was originally loaded from the disk.

Syntax: [ClipboardGetBitmap\(ListImageName\\$,Result\)](#)

Parameters:

[ListImageName\\$](#) An image to be created in the PiXCL image list. This can be any name, but should preferably include a path if you want to save it, and specify one of the supported bitmap formats as the extension. Note that this format will only be effective once the bitmap has been saved to disk.

[Result](#) 1 if the operation was successful, otherwise 0. If [ListImageName\\$](#) is in the list, it is first deleted and the new image replaces it.

Remarks:

If you are working with RLE images, please be aware of the following:

1. An RLE4 image loaded from the disk will be rendered as 8 bits per pixel in the image list. PiXCL support saving RLE images in 8-bit mode only.
2. You can copy a 4-bit image from an image editor to the clipboard, and then to the PiXCL image list, and it will be in 4-bits per pixel.
3. Images are stored in the image list as UNCOMPRESSED bitmaps, even if the source image was compressed.

Related Commands:

[ClipboardEmpty](#) [ClipboardPutBitmap](#) [CopyWindowToClipboard](#)

ClipboardPut

Copies text to the Windows Clipboard, replacing the Clipboard's current contents.

Syntax: `ClipboardPut(String,$,Result)`

Parameters:

String\$ A string containing the text you want to copy to the Clipboard.

Result An integer variable that indicates the outcome of the Clipboard operation. If the operation was successful, this variable is assigned a value of 1. If there was not enough memory to carry out the operation, this variable is assigned a value of 0.

Remarks:

PiXCL always clears the Clipboard before copying text to it.

If you want text to appear on separate lines in the Clipboard, be sure to place a carriage return and a linefeed character at the end of each line (see the example).

Example:

This example places a carriage return and linefeed character between the words "Testing" and "123". It then copies the result to the Clipboard.

```
{Place carriage return in CR$ and linefeed in LF$}
```

```
  Chr(13,CR$)
```

```
  Chr(10,LF$)
```

```
{Put carriage return/linefeed between two strings}
```

```
  Toclip$ = "Testing" + CR$
```

```
  Toclip$ = Toclip$ + LF$
```

```
  Toclip$ = Toclip$ + "123"
```

```
{Copy Contents$ to the Clipboard}
```

```
  ClipboardPut(Toclip$,Result)
```

Related Commands:

[ClipboardEmpty](#) [ClipboardGet](#) [ClipboardAppend](#)

ClipboardPutBitmap

This command takes an image that is loaded in the PiXCL imagelist and passes a **COPY** of it into the Clipboard. All images loaded into PiXCL are Windows bitmaps, regardless of the type of image that was originally loaded from the disk.

Syntax: [ClipboardPutBitmap\(ListImageName\\$,Result\)](#)

Parameters:

[ListImageName\\$](#) An image that has been loaded into PiXCL.
[Result](#) 1 if the operation was successful, otherwise 0. If [ListImageName\\$](#) is not in the list, the operation fails.

Remarks:

It is possible to copy a window to the Clipboard and then copy it back to the PiXCL image list.

Related Commands:

[ClipboardGetBitmap](#) [CopyWindowToClipboard](#)

ClipCursor

PIXCL 5 command. The [ClipCursor](#) function confines the cursor to a rectangular area on the virtual screen. If a subsequent cursor position lies outside the rectangle, the system automatically adjusts the position to keep the cursor inside the rectangular area. The default is the coordinates of the primary monitor screen (e.g. 0,0,1024,768).

Syntax: [ClipCursor\(sx1,sy1,sx2,sy2\)](#)

Parameters:

[sx1,sy1,sx2,sy2](#) The screen coordinates for the desired region.

Remarks:

The cursor is a GLOBAL resource, so unless you want to leave the clip region active for all other applications, your PIXCL application must include [ClipCursor\(0,0,0,0\)](#) to reset the default.

Related Commands:

[GetClipCursor](#)

CloseBitmapWindow

Use this command to close a bitmap window. Closing a bitmap window does NOT delete the bitmap from the PiXCL image list. This requires the [FreeBitmap](#) command.

Syntax: [CloseBitmapWindow\(WindowID\)](#)

Parameters:

[WindowID](#) The ID number **variable** returned from the [DrawBitmapWindow](#) command. [WindowID](#) returns a zero if the bitmap window was closed.

Remarks:

To close all bitmap windows, set [WindowID](#) to zero. A common programming error is to use a value for [WindowID](#) instead of a variable. E.g.

`CloseBitmapWindow(0)` will cause a syntax error, whereas

`WindowID = 0`

`CloseBitmapWindow(WindowID)` will run correctly.

Related Commands:

[DrawBitmapWindow](#) [FlashBMWindow](#) [SetBMWMouse](#) [ZoomBitmapWindow](#) [FreeBitmap](#) [FreeBitmapAll](#)

CloseEvent

Events may be used or required by extension libraries only. For more information, see the topics on Synchronization Functions in your compiler documentation. This function terminates an existing event.

Syntax: [CloseEvent\(EventHandle,Result\)](#)

Parameters:

[EventHandle](#) The event handle created by [CreateEvent](#).
[Result](#) 1 if the function succeeds, otherwise 0.

Related Commands:

[CreateEvent](#) [PulseEvent](#) [ResetEvent](#)

CmdLineArg

You can get individual command line arguments and pass them into string variables with the [CmdLineArg](#) command. This saves you the effort of parsing the command line yourself in PIXCL code. Arguments should be delimited with a space character.

Syntax: [CmdLineArg\(Index, Arg\\$\)](#)

Parameters:

[Index](#) Starting from 1, the command line argument.

[Arg\\$](#) The argument. If Index is greater than the number of arguments, the return string is empty.

Related Commands:

[GetCommandLine](#) [CountCmdLineArgs](#)

ComboBox

A combo box is a unique type of control that combines much of the functionality of a list box and an edit control.

Comboboxes provide three styles, the SIMPLE combobox, which has an edit/display region with a permanently enabled dropdown region; or

or DROPDOWN, which has an edit control with a dropdown arrow control;

or DROPDOWNLIST, generally used when the string variable is a list.

A combo box consists of a list and a selection field. The list presents the options a user can select and the selection field displays the current selection. Except in drop-down list boxes, the selection field is an edit control and can be used to enter text not in the list.

Syntax: `ComboBox()` to clear all comboboxes, or
`ComboBox(x1,y1,x2,y2,TOKEN,List$,Delimiter$,Input$...)`

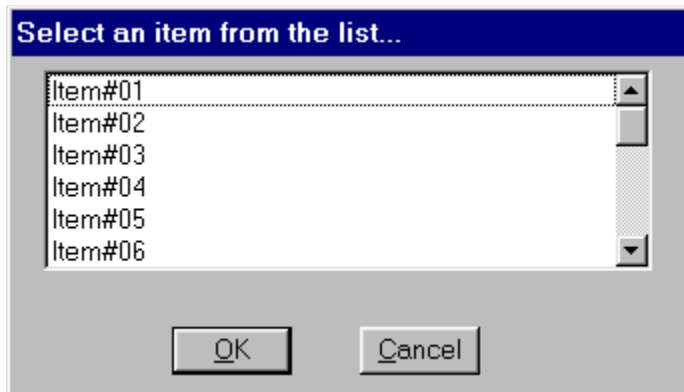
Parameters:

| | |
|---------------------------|---|
| <code>x1,y1,x2,y2</code> | The coordinates of the combobox. Note that this includes the dropdown region, if appropriate, as well. |
| <code>SIMPLE</code> | Creates a simple combobox with a permanent listing region below it. |
| <code>DROPDOWN</code> | Creates a combobox with a down arrow that makes a dropdown region. A list item can be selected, or you can type in a new selection into the edit control. |
| <code>DROPDOWNLIST</code> | Creates a combobox with a down arrow that makes a dropdown region. The user input function is disabled, and you can only select one of the items in the list. |
| <code>List\$</code> | The optional list that appears in the dropdown region. |
| <code>Delimiter\$</code> | The single character (e.g. " ") that is used to delimit each item in the list. This character must follow all list items. If you want a list item to be a null string, use two delimiters e.g. " ". If the last item does not have a delimiter character, it will not appear in the dropdown list. |
| <code>Input\$</code> | The default variable that is used to display the current selection and return the selected or entered string. |

Remarks:

The text that appears in the combobox is written in the current font, in monochrome, regardless of the current font color. For SIMPLE and DROPDOWN styles, the edit control will automatically scroll horizontally if the text will not fit in the available space.

For DROPDOWN and DROPDOWNLIST styles, a vertical scrollbar is created if the list is longer than the available dropdown window.



Multiple comboboxes styles and button.

Example:

The above figure was produced with the code fragment in this example. Note the vertical scrollbar in the lefthand control. Once input is complete, the "Done" button is clicked to update the return strings, and draw them in the client area. Note also that the final list item ("CBarg#8") does not appear because the delimiter character has been omitted.

```

Make_Combobox:
  DrawBackground
  UseFont("MS Sans Serif",0,19,NOBOLD,NOITALIC,NOUNDERLINE,0,255,0)
  Cbox1$ = "ComboBox#1"
  Cbox2$ = "ComboBox#2"
  Cbox3$ = "ComboBox#3"
  Delimiter$ = "|"
  List1$ = "CBarg#1|CBarg#2|CBarg#3|CBarg#4|CBarg#5|CBarg#6|CBarg#7|CBarg#8"

  List2$ = "CBarg#1|CBarg#2|CBarg#3|CBarg#4|"

  ComboBox(5,10,120,145,DROPDOWN,List1$,Delimiter$,Cbox1$,
    130,10,230,180,DROPDOWNLIST,List2$,Delimiter$,Cbox2$,
    245,10,345,95,SIMPLE,List1$,Delimiter$,Cbox3$)
  WaitInput(1)
  Button(360,10,420,45,PUSH,"Done",Remove_Combobox)
  WaitInput()

Remove_Combobox:
  DrawBackground
  Button()
  ComboBox()
  UseFont("MS Sans Serif",0,19,BOLD,NOITALIC,NOUNDERLINE,0,255,0)
  DrawText(10,10,Cbox1$)
  DrawText(10,30,Cbox2$)
  DrawText(10,50,Cbox3$)
  Goto Wait_for_Input

```

Related Commands:

[Button](#) [DialogBox](#) [SetEditControl](#)

CopyArray

PiXCL 5 command. The contents or partial contents of an array can be copied to another array of the same type. You can also copy one portion of an array to another portion of the same array.

Syntax: `CopyArray(SrcArray[src_start_index], DstArray[dst_start_index], CopyElements, Result)`

Parameters:

| | |
|------------------------|--|
| <i>SrcArray</i> | The name of the source array. |
| <i>src_start_index</i> | The source array start index from which elements are copied. |
| <i>DstArray</i> | The name of the destination array. |
| <i>dst_start_index</i> | The destination array start index to which elements are copied. |
| <i>CopyElements</i> | The number of elements to copy. |
| <i>Result</i> | The number of elements copied, otherwise 0. The command will fail if the indices are out of range. |

Related Commands:

[Array](#) [FreeArrayVar](#) [FreeArrayVarAll](#) [RestoreArray](#) [SaveArray](#)

CopyWindowToClipboard

Any top-level window, including a current PiXCL application and most child windows, can have a copy of that window pasted to the Clipboard as a bitmap. This is often called window capturing or grabbing. Once in the Clipboard, the image can be pasted into any other application that supports bitmap cut and paste. Examples are MS-Word, JASC Inc PaintShopPro, Adobe PhotoShop, and any other image editing application. The pasted bitmap will always be 24 bits per pixel. You may need to reduce the pixel depth of the pasted image using facilities in your image editor.

Syntax: [CopyWindowToClipboard\(*TopLevelWindow*,\\$,*ChildWindow*,\\$,*Result*\)](#)

Parameters:

[TopLevelWindow](#)\$ The name that appears in the target window titlebar. This is also used for POPUP windows such as toolbars and dialog boxes.

[ChildWindow](#)\$ The name of a child window of [TopLevelWindow](#)\$. Set this to a NULL string to get a top-level window. Child windows may not always be accessible for transfer.

[Result](#) 1 if the operation was successful, otherwise 0.

Remarks:

The Clipboard is emptied of all other data before the transfer occurs. It is your responsibility to empty the Clipboard once the window snapshot is no longer required. To paste a bitmap into the PiXCL image list, use the [ClipboardGetBitmap](#) command.

Related Commands:

[ClipboardEmpty](#) [ClipboardGetBitmap](#) [EnumWindows](#) [EnumChildWindows](#)

Cos

Floating Point math library function. Calculate the cosine of an angle in radians.

Syntax: `Cos(Angle&, Value&)`

Parameters:

`Angle&` The angle in radians
`Value&` The result of the function.

Related Commands:

[Sin, Tan](#)

Cosh

Floating Point math library function. Calculate the hyperbolic cosine of an angle in radians.

Syntax: `Cosh(Angle&, Value&)`

Parameters:

`Angle&` The angle in radians
`Value&` The result of the function.

Related Commands:

[Sinh Tan](#)

ConvertPALfile

geoPiXCL command only. This command complements the [CreatePALfile](#) command, and converts a PiXCL/geoPiXCL ascii PAL file into a binary PBL file that includes the same information, and vice-versa.

Syntax: `ConvertPALfile(PaletteFile$,Res)`

Parameters:

PaletteFile\$ A valid PiXCL PAL or PBL file.
Result 1 if the operation was successful, otherwise 0. If the files are not valid ascii or binary files, the operation will return 0.

Related Command:

[CreatePALfile](#)

CopyListBitmapRect

Create a subarea of an image loaded into memory. This can be useful where you need to get a subarea of a compressed image e.g. JPEG.

Syntax: `CopyListBitmapRect(x1,y1,x2,y2,ListImageName$,SubAreaName$,Result)`

Parameters:

| | |
|------------------------|--|
| <i>x1,y1,x2,y2</i> | The sub-area coordinates. |
| <i>ListImageName\$</i> | The source image loaded into the PiXCL image list. |
| <i>SubAreaName\$</i> | The name of the new subarea. |
| <i>Result</i> | 1 if the command was successful, otherwise 0. |

Related Commands:

CountBitmapColors

This command returns the number of unique colors in any of the supported bitmap formats. The algorithm treats each pixel in the image as an integer and creates a dynamically sorted linked list as it scans through the bitmap. PiXCL creates a zero filled bit array that corresponds to the number of possible colors. Hence for a 24 bit image, this array is 2MB in size. Each new pixel value is the bit address in the array, and the bit is set to 1. At the end of the process, the number of unique colors is the number of '1' bits, which is returned to the command, and the bit array memory is returned to the operating system.

If the specified bitmap *Filename\$* has been loaded into the PiXCL Bitmap List, [CountBitmapColors](#) accesses the memory copy, otherwise the bitmap *Filename\$* is read from the disk.

Syntax: [CountBitmapColors](#)(*Filename\$,Colors*)

Parameters:

Filename\$ The desired image filename string.

Colors The number of unique colors. This will be a number in the range 1 to the number of pixels in the image, or the maximum number of colors possible in the bitmap format, whichever is the smaller. If the function fails, for example, if the file cannot be found or is an unrecognised format, *Colors* returns 0.

Related Commands:

[GetBitmapDim](#)

CountCmdLineArgs

If you invoke a PiXCL application with a command line that has a variable number of arguments, [CountCmdLineArgs](#) returns the number of arguments by counting the space delimiters which will always be present.

Syntax: [CountCmdLineArgs\(ArgNumber\)](#)

Parameter:

[ArgNumber](#) The number of arguments after the EXE name.

Related Command:

[GetCmdLine](#)

CountRegdUserCmds

The number of registered external commands can be obtained with this command.

Syntax: [CountRegdUserCmds\(Count\)](#)

Parameter:

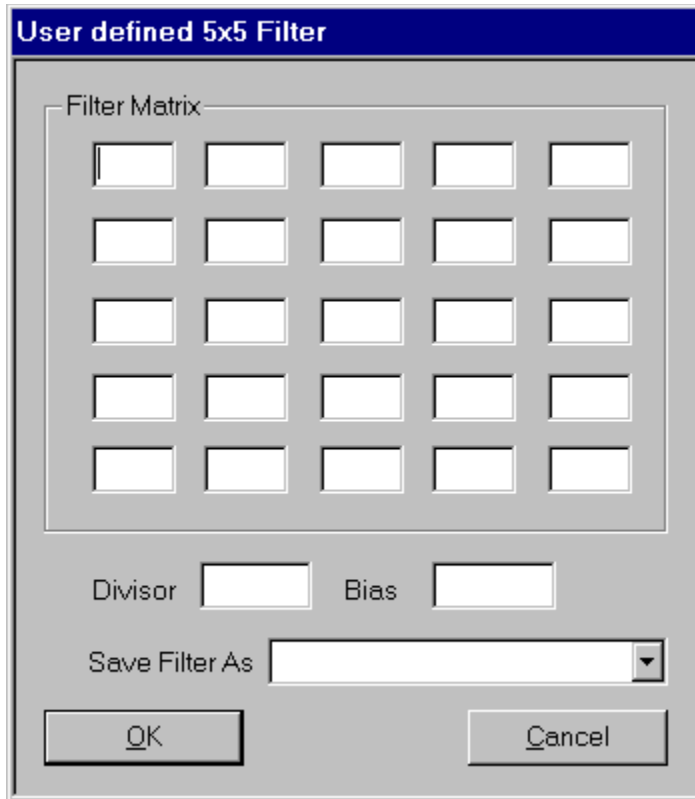
[Count](#) The number of registered commands.

Related Commands:

[RegisterExtLibCmdSet](#) [RegisterUserCommand](#) [UnregisterUserCmds](#)

Create5x5Filter

PiXCL supports 5x5 and 15x15 filters. You can display a dialog box to edit an existing 5x5 filter or create a new 5x5 filter. When you need to create 15x15 filters, we suggest the PiXCL MDI Editor, or another text editor.



Syntax: `Create5x5Filter(Filter$,Result)`

Parameters:

Filter\$

The name of the filter. If the filter is invalid in some way or does not exist, a MessageBox informs you of the problem, then displays the empty dialog, above.

Result

1 if the operation has successfully written a new filter file, otherwise 0. Returns 0 if the Cancel button is pressed.

Related Commands:

[FilterImage5x](#), [ViewFilterFile](#)

CreateBitmap

A memory bitmap of arbitrary dimensions and pixel depth can be created in the PIXCL image list, as either a 256 colour paletted image with a user-defined colormap, or as a full colour 24-bit image. The bitmap data is set to black (0,0,0). The bitmaps created with [CreateBitmap](#) will generally be saved to disk with the [SaveBitmap](#) command. Once the new bitmap has been created in memory, it is set as the current bitmap for image processing or save operations.

Syntax: [CreateBitmap](#)(*Pixels*, *Lines*, *Bits*, *PaletteFile\$*, *ImageName\$*, *Result*)

Parameters:

| | |
|--|--|
| Pixels , Lines | The dimensions of the new bitmap. |
| Bits | The pixel depth desired. Must be either 8 or 24 or 32. Any other value will cause the command to fail and Result returns 0. |
| PaletteFile\$ | The full path and name of an ascii PAL file. See RemapImage for details of the PAL file format. PaletteFile\$ is ignored if Bits is 24 or 32. You can also import ascii palette files created in JASC Paint Shop Pro 4, 5 and 6. |
| ImageName\$ | The name of the new bitmap added to the image list. If an image of the same name is already in the list, it is first deleted, and the new image added. |
| Result | 1 if the operation succeeded. If the palette file format is not recognized, Result returns 0. If the palette file is not found, Result returns -1. |

Related Commands:

[CreatePalFile](#) [RemapImage](#) [SaveBitmap](#) [CopyListBitmapRect](#)

CreateColorSpace

PIXCL 5 command. A colour space is a mapping of colour components onto a Cartesian coordinate system in three or more dimensions.

Syntax: `CreateColorSpace(Type_TOKEN, Intent_TOKEN, RedX,RedY,RedZ, GreenX,GreenY,GreenZ, BlueZ,BlueY,BlueZ, RGamma,GGamma,BGamma, CSHandle)`

Parameters:

| | |
|---------------------|--|
| Type_TOKEN | CALIBRATED_RGB Color values are calibrated RGB values. The values are translated using the endpoints. WINDOWS_COLOR_SPACE Color values are Windows default color space color values. sRGB Use sRGB values. |
| Intent_TOKEN | ABS_COLORIMETRIC Maintain the white point. Match the colors to their nearest color in the destination gamut. BUSINESS Maintain saturation. Used for business charts and other situations in which undithered colors are required. GRAPHICS Maintain colorimetric match. Used for graphic designs and named colors. IMAGES Maintain contrast. Used for photographs and natural images. |
| <i>RedX Y Z</i> | The red endpoints. |
| <i>GreenX Y Z</i> | The green endpoints. |
| <i>BlueX Y Z</i> | The blue endpoints. |
| <i>R G BGamma</i> | The gamma scaling values (x100?) |
| <i>CSHandle</i> | The handle of the created colour space, otherwise 0. |

Related Commands:

DeleteColorSpace

CreateCOMInstance

PiXCL 5 command: subject to change. A PiXCL application can become a COM client to a COM server. Calling this command initializes the Windows COM libraries. When using PiXCL as a COM client, you will need to have the necessary access or programming or scripting information on the server application.

Uses of COM include Automation of servers (eg MS-Office applications) and ActiveX controls.

Syntax: `CreateCOMInstance(ProgramID$, ServerMode_TOKEN, Computer$, IUnknown)`

Parameters:

| | |
|-------------------------------|--|
| <code>ProgramID\$</code> | The ProgramID string that identifies the server application. The registry is accessed and the classID and GUID acquired. More notes required. |
| <code>ServerMode_TOKEN</code> | One of the following: INPROC_SERVER if the server is a DLL on the local machine. The DLL becomes part of the PiXCL process. This is known as in-process mode. INPROC_HANDLER if this is a DLL that runs in the client process and implements client-side structures of this class when instances of the class are accessed remotely LOCAL_SERVER if the server is a EXE on the local machine. That is, it runs on same machine but in a different process. Paint Shop Pro 4, 5 and 6, MS-Word are examples. This will likely be the most commonly used mode. REMOTE_SERVER if the server is a EXE on a remote machine. CTX_SERVER indicates server code, whether in-process, local, or remote. CTX_ALL indicates all class contexts. |
| <code>Computer\$</code> | For the local machine INPROC or LOCAL servers, use "". When specifying a remote machine, use the naming scheme of the network transport link. By default, all UNC ("\\server" or "server") and DNS names ("server.com", "www.foo.com", or "135.5.33.19") names are allowed. |
| <code>IUnknown</code> | If the command is successful, returns a non-zero number that is the pointer to the server IUnknown interface, otherwise 0. For Microsoft servers, these numbers will be positive, and for third-party servers, negative. |

Remarks:

It is advisable that ReleaseCOM be called before your PiXCL application exits.

Related Commands:

[ReleaseCOM](#) QueryCOMInstance

CreateEvent

Events may be used or required by extension libraries only. For more information, see the topics on Synchronization Functions in your compiler documentation.

Syntax: [CreateEvent\(ManualReset, InitialState, Name\\$, EventHandle\)](#)

Parameters:

| | |
|------------------------------|--|
| ManualReset | Specifies whether a manual-reset or auto-reset event object is created. If @TRUE, then you must use the ResetEvent function to manually reset the state to nonsignaled. If @FALSE, the system automatically resets the state to nonsignaled after a single waiting thread has been released. |
| InitialState | Specifies the initial state of the event object. If TRUE, the initial state is signaled; otherwise, it is nonsignaled. |
| Name\$ | A name unique to your program, or a null string. |
| EventHandle | The event handle that is used by the other event commands. If the function fails, EventHandle returns 0. |

Related Commands:

[CloseEvent](#) [PulseEvent](#) [ResetEvent](#)

CreatePALfile

Palette files are ASCII files used with the [RemapImage](#) command and other commands where a set of 256 palette values are required.

Syntax: `CreatePALfile(Filename$,MODE_token,Result,
[NumberOfClasses, class range pair triplets])`

Parameters:

| | |
|----------------------------|--|
| Filename\$ | The name of the PAL file to be created. If the file exists, it is overwritten. If you need to keep an existing file, use the FileExist command first. |
| STANDARD | Creates a standard PAL file with entries from “0 0 0” to “255 255 255”. Applying this file to an image will not change it. |
| INVERSE | Creates an inverse PAL file with entries from “255 255 255” to “0 0 0”. Applying this file to an image will invert the colors. It is functionally equivalent to using the InvertImage command. |
| CLASSIFY | A classification PAL file is to be created, according to the notes below. |
| Result | 1 if the operation succeeded, otherwise zero. For STANDARD and INVERSE modes, this is the last argument. |

For [CLASSIFY](#) mode only ...

| | |
|-----------------------------------|---|
| NumberOfClasses | The number of class range pairs following. The maximum is 8 sets. |
| RLoValue, RHValue | The range of values for each class in the Red channel. |
| GLoValue, GHValue | The range of values for each class in the Green channel. |
| BLoValue, BHValue | The range of values for each class in the Blue channel. |

Remarks:

[LoValues](#) must be less than [HiValues](#) in each channel, and classes must be defined so that they do not overlap on previous classes. E.g Class#1 5-13; Class#2 17-32 Class#3 41-55 and so on. If any overlap exists, [Result](#) returns 0 and no file is created.

[CreatePALfile](#) is a variable argument command, so only the range values desired are required. The colors used for each class are predefined as **Red, Green, Blue, Yellow, Magenta, Cyan, Orange** and **Mid-Gray**. These colors can be changed by using the [CustomColor](#) command prior to [CreatePALfile](#) to set the desired class colors.

Related Commands:

[ConvertPALfile](#) [CustomColor](#) [InvertImage](#) [RemapImage](#)

CustomColor

This command goes with the [ChooseColor](#) command that displays a common dialog with 16 user definable colors (initially set to black), and these can be set or reset to any desired color, while the PIXCL application is running. Hence, if your application needs to define a specific set of colors for pens, brushes or backgrounds, or perhaps palette file entries, you must program these into the application, or possibly read them from an INI file, or from a previously defined location in the Registry.

Syntax: [CustomColor\(r1,g1,b1, r2,g3,b3,....r16,g16,b16\)](#)

Parameters:

[r,g,b](#)

Integer color values. These must be in the range 0-255. If they are not they are clipped by ANDing the value with 0x000000FF. It is preferable but not essential that your script ensures that the color ranges are correct. All sixteen colors must be defined, even if set to 0,0,0.

Related Commands:

[CreatePALfile](#) [ChooseColor](#) [UseBackground](#) [UsePen](#) [UseBrush](#)

CustomizeToolBtn

Toolbars and toolwindows can be customized by moving and deleting buttons. A system dialog can be displayed either by double clicking the toolbar or toolwindow background, or by issuing the [CustomizeToolBtn](#) command.

Syntax: [CustomizeToolBtn](#)(*ToolWindow\$*)

Parameters:

[ToolWindow\\$](#) For the toolbar, set this to a NULL string (""). For a toolwindow, use the relevant name specified in the [ToolWindow](#) command.

Related Commands:

[ChangeToolbarBtn](#) [GetToolBarBtnStatus](#) [ToolBar](#) [ToolWindow](#)

DDEExecute

Sends a command string to a server application in a Dynamic Data Exchange (DDE) conversation.

Syntax: `DDEExecute(ChanNum,ExecuteString$,Result)`

Parameters:

| | |
|------------------------------|---|
| <code>ChanNum</code> | The channel number established by the DDEInitiate command. |
| <code>ExecuteString\$</code> | A string containing the command to execute. The command must match the format that the server application will recognize, usually in the application's native macro language. |
| <code>Result</code> | An integer variable that indicates the outcome of the transaction. It is assigned 1 if the transaction was successful; otherwise 0. |

Remarks:

Before using DDEExecute, you must use a DDEInitiate command to start a DDE conversation with a server application and establish a valid value for the `ChanNum` variable.

Each server application requires its own format for the command strings that are sent to it in a DDE conversation. Typically, the format must match the application's native macro language. For example, when sending a command to Microsoft Excel, it must be in Excel's macro language (see the second example). Another common requirement is that the command string be enclosed in square brackets.

Not all Windows applications support DDE, which has been largely superceded by OLE 2 (Object Linking and Embedding, version 2) software technology. Some applications, mostly from Microsoft, do support DDE communications. The best way to find out is to try some of the tests here.

You can also search in the Registry for a **ddeexec** key related to the application of interest. Subkeys **application** and **topic** are the DDE Service name and main Topic respectively. See the [DDEInitiate](#) topic for more information.

Examples:

The following example shows how to use the DDEExecute command to activate the Main program group in Program Manager.

```
{Initiate DDE conversation with Program Manager}
  DDEInitiate ("PROGMAN", "PROGMAN", ChanNum)

{Set command to execute. Because Main is a personal group,
 ShowGroup's third parameter is set to 0. (With Common
 program groups, third parameter is set to 1 or left out.)}
  ExecuteString$ = "[ShowGroup(Main,1,0)]"

{Send the execute message}
  DDEExecute(ChanNum,ExecuteString$,Result)
  If Result = 0 Then Beep

{Terminate the conversation}
  DDETerminate(ChanNum)
```

Windows Explorer in Win95/NT4 also supports DDE conversations. You can bring up the FileFind dialog, and Explore or View a specific folder contents.

```
Start_Explorer_DDE:
  DDEInitiate("Folders","AppProperties",ChanNum)
  Goto Wait_for_Input
```



```

FindFolder_DDE:
{display the Find: All Files dialog from Explorer.
  DDEexec = [FindFolder("%1", %I)]
  arg#1 = the start-in [path]directory string,
          no spaces; quotes not needed
  arg#2 = number. Set to 0.
}
UseCursor(WAIT)
DDEExecute(ChanNum,"[FindFolder(c:\windows\system,0)]",Res)
If Res = 0 Then DrawText(10,25,"DDE Exec failed.")
UseCursor(ARROW)
Goto Wait_for_Input

ExploreFolder_DDE:
{display the Find: All Files dialog from explorer.
  arg#1 = [path]directory;
  arg#2 = unknown, set to 0.
  arg#3 = 0 | 1 -> Explorer window not visible | visible
}
UseCursor(WAIT)
DDEExecute(ChanNum,"[ExploreFolder(c:\windows\system, 0, 1)]",Res)
If Res = 0 Then DrawText(10,25,"DDE Exec failed.")
UseCursor(ARROW)
Goto Wait_for_Input

ViewFolder_DDE: {display the Find: All Files dialog from explorer. }
{arg#1 = [path]directory;
  arg#2 = set to 0
  arg#3 = 0 | 1 ->Explorer not visible | visible
}
UseCursor(WAIT)
DDEExecute(ChanNum,"[ViewFolder(c:\windows\system, 0, 1)]",Res)
If Res = 0 Then DrawText(10,25,"DDE Exec failed.")
UseCursor(ARROW)
Goto Wait_for_Input

FindFile_DDE:
{display the Find: All Files dialog from Explorer.
  arg#1 = start-in directory string, has to be valid
          directory eg \ on the C: drive. Don't seem
          able to set the start disk to any other disk.
  arg#2 = set to 0
}
UseCursor(WAIT)
DDEExecute(ChanNum,"[OpenFindFile(\,0)]",Res)
If Res = 0 Then DrawText(10,25,"DDE Exec failed.")
UseCursor(ARROW)
Goto Wait_for_Input

```

The next example shows how to start a DDE conversation with Microsoft Excel 97 using the "System" topic. This allows **DDEExecute** to send Excel a command string to execute in the form of an Excel macro function. Here the example instructs Excel to open the worksheet "SALES.XLS". (The example assumes that Excel is already running.)

```
{Initiate DDE conversation}
```

```
DDEInitiate("Excel","System",ChanNum)

(Create command string that uses double quotes in the form
[OPEN("C:\EXCEL\EXAMPLES\BUDGET.XLS")])
Chr(34,DbQuote$)
Command$ = "[OPEN("
Command$ = Command$ + DbQuote$
Command$ = Command$ + "C:\EXCEL\EXAMPLES\BUDGET.XLS"
Command$ = Command$ + DbQuote$
Command$ = Command$ + "]"

(Send the command string)
DDEExecute(ChanNum,Command$,Result)
If Result = 0 Then Beep

(Terminate the conversation)
DDETerminate(ChanNum)
```

Related Commands:

[DDEInitiate](#) [DDETerminate](#) [DDEPoke](#) and [DDERequest](#)

DDEInitiate

Initiates a new DDE conversation and returns the associated channel number.

Syntax: `DDEInitiate(Service$,Topic$,ChanNum)`

Parameters:

| | |
|------------------------|--|
| <code>Service\$</code> | A string that specifies the service name of the server application with which a conversation is to be established--for example, "Excel". |
| <code>Topic\$</code> | A string that specifies the name of the topic on which a conversation is to be established--for example, "System". |
| <code>ChanNum</code> | A numeric variable that will contain the channel number of the established conversation. If the function fails, <code>ChanNum</code> is assigned a value of 0. |

Remarks:

PiXCL can function only as a client to a DDE server (such as Windows Explorer) in a DDE conversation.

The service name of a server application usually matches its program name, and can be found in the Registry. Look for a **ddeexec** key in the application of interest entries. Subkeys **application** and **topic** are the Service Name and main Topic respectively.

For example, Service Names and some topics are ...

"**Excel**" for Microsoft Excel 97. Multiple topics exist.

"**MSAccess**" for Microsoft Access 97. Multiple topics exist.

"**WinWord**" for Microsoft Word 97. Multiple topics exist.

"**Folders**" for Microsoft Windows Explorer, only topic is "AppProperties".

"**IExplore**" for Microsoft Internet Explorer, only topic is "WWW_OpenURL".

"**Netscape**" for Netscape Navigator, only topic is "WWW_OpenURL".

The topic name is a string that identifies the topic for the conversation. Each application has its own set of valid topic names. With many applications, the name of an open file is a valid topic name. For example, if Excel is currently using the the worksheet file "SALES.XLS," you could use the following PiXCL command to initiate a DDE conversation with that Excel worksheet:

```
DDEInitiate("Excel","C:\SALES.XLS",ChanNum)
```

By using the "**System**" topic, you can often use the [DDERequest](#) command to obtain a list of the strings that an application accepts as valid topic names--for example, a list of the files that are currently in use. (See the [DDERequest](#) command more explanation and an example using Excel.)

PiXCL can set up multiple, simultaneous DDE conversations. Your program must keep track of the DDE channel created, and ensure that they are correctly terminated.

Once a DDE conversation has been initiated with a server application, you can use any of these DDE-related commands to communicate with it:

- [DDERequest](#) -- To read information from the server.
- [DDEPoke](#) -- To write information to the server.
- [DDEExecute](#) -- To send a command string to be executed by the server.

Example:

See the [DDEExecute](#), [DDEPoke](#), and [DDERequest](#) commands for examples.

Related Commands:

[DDETerminate](#) [DDEPoke](#) [DDERequest](#) [DDEExecute](#)

DDEPoke

Sends data to an item in a DDE conversation. You can for example send data an Excel spreadsheet cell.

Syntax: `DDEPoke(ChanNum,Item$,Data$,Result)`

Parameters:

| | |
|----------------|---|
| <i>ChanNum</i> | The channel number established by the DDEInitiate command. |
| <i>Item\$</i> | A string that identifies an item that is appropriate for the server application. Different applications support different item names. In an Excel worksheet, for example, you can use a string of the form "R1C1" to identify row 1, column 1 in the worksheet. |
| <i>Data\$</i> | A string containing the data you want to send to the item identified in <i>Item\$</i> . |
| <i>Result</i> | An integer variable that indicates the outcome of the transaction. It is assigned 1 if the transaction was successful; otherwise 0. |

Example:

The following example stores the simple message "Hello from PiXCL!" in the first cell of an Excel worksheet. This example assumes that you've just started Excel and that Sheet1 is the active worksheet:

```
DDEInitiate("EXCEL", "[Book1]Sheet1", ChanNum)
DDEPoke(ChanNum, "R1C1", "Hello from PiXCL!", Ignore)
DDETerminate(ChanNum)
```

Related Commands:

[DDEInitiate](#) [DDETerminate](#) [DDERequest](#) [DDEExecute](#)

DDERequest

Requests information from a specified item in a DDE conversation.

Syntax: `DDERequest(ChanNum,Item$,Result$)`

Parameters:

- ChanNum** The channel number established by the `DDEInitiate` command.
- Item\$** A string that identifies an item that is appropriate for the server application. Different applications support different item names.
- Result\$** A string of information returned from the item. If the request is unsuccessful, a null string (" ") is returned.

Remarks:

Each application supports its own set of topic and item names. Many, but not all, DDE server applications support the "System" topic. If it does, it will usually support one or more of the following topics as well:

"Formats" "Status" "SysItems" "Topics" "Help" "ReturnMessage" "TopicItemList"

For example, The Table below shows some of the items that you can use with `DDERequest` after you've initiated a conversation with Excel using a given topic name.

| Topic name | Item name | Result\$ returns |
|---|---|--|
| "System" | "SysItems" | A list of item names that you can use with the "System" topic in subsequent DDE commands, shown below. |
| | "Topics" | A list of DDE topic names accepted by Excel, including the names of all open worksheet files. |
| | "Formats" | A list of Clipboard formats supported by Excel--for example, "TEXT BITMAP". |
| | "Status" | Returns "Ready" if the spreadsheet can be accessed. |
| | "Selection" | The currently selected cell e.g. "[Book1]Sheet1!R1C1" |
| | Protocols | Returns "StdFileEditing" and "Embedding". These are used with OLE functions. |
| | EditEnvItems | Returns "StdHostNames", "StdTargetDevice" and "StdDocDimensions". |
| The name of an Excel worksheet currently in use, e.g. "[Book1]Sheet1" | A string of the form "R1C1" (for row 1, column 1) | A string representing the current contents of the specified cell. |

Examples:

The following example uses `DDERequest` to get the current contents of the first cell in an Excel worksheet and display it in the PIXCL window. This example assumes Excel is already running.

```
DDEInitiate("EXCEL", "Sheet1", ChanNum)
DDERequest(ChanNum, "R1C1", Contents$)
```

```
DDETerminate(ChanNum)
DrawText(10,10,Contents$)
WaitInput()
```

The next example gets all the group names from Program Manager and draws the string that contains them on the screen.

```
Item$ = "Groups" {Get all group names from Program Manager}
DDEInitiate("PROGMAN","PROGMAN",ChanNum)
DDERequest(ChanNum,Item$,List$)
DrawText(10,10,List$)
DDETerminate(ChanNum)
WaitInput()
```

If you try this example, you'll see that the names are delimited by carriage return / linefeed pairs.

The next example uses the "System" topic with the "SysItems" item to get a list of available item names from Excel. A list box displays the results. This example assumes Excel is already running.

```
DDEInitiate("Excel","System",ChanNum)
DDERequest(ChanNum,"SysItems",List$)
DDETerminate(ChanNum)
Chr(9,Tab$) {List is delimited by Tabs}
ListBox("Excel System topics",List$,Tab$,Ignore$)
WaitInput()
```

This final example sends a word to Winword for spell checking. It shows how to use some of the available DDE commands. See your Word and WordBasic documentation for more information.

```
{Initiate DDE conversation}
  DDEInitiate("Winword","Document1",ChanNum)

{Build bookmark command}
  Item$ = "here" {This is just a placeholder}
  Chr(34,DbQuote$)
  ExecuteString$ = "[InsertBookMark.Name = "
  ExecuteString$ = ExecuteString$ + DbQuote$
  ExecuteString$ = ExecuteString$ + Item$
  ExecuteString$ = ExecuteString$ + DbQuote$
  ExecuteString$ = ExecuteString$ + "]"

{Create bookmark in Word}
  DDEExecute(ChanNum,ExecuteString$,Result)
  If Result = 0 Then Beep

{This the word to check}
WordCheck$ = "Testiing"

{Send WordCheck$ to bookmark}
  DDEPoke(ChanNum,Item$,WordCheck$,Result)
  If Result = 0 Then Beep

{Spell check it}
  DDEExecute(ChanNum,"[ToolsSpellSelection]",Result)

{Put up button for requesting word back from Word}
  UseCoordinates(PIXEL)
  Button(200,200,400,300,"Request back",Request)
```

```
WaitInput()

{Request word back from Word and display it on screen}
Request:
  DDERequest(ChanNum,Item$,Result$)
  If Result$ <> "" Then WordCheck$ = Result$
  UseCoordinates(METRIC)
  DrawText(10,50,WordCheck$)

Wait_for_input:
  DDETerminate(ChanNum)
  WaitInput()
```

See also the sample program **pxexplor.pxl** which demonstrates DDE conversations with Explorer, Access, Excel and Netscape or Internet Explorer.

Related Commands:

[DDEInitiate](#) [DDETerminate](#) [DDEPoke](#) [DDEExecute](#)

DDETerminate

Closes a DDE conversation.

Syntax: [DDETerminate\(ChanNum\)](#)

Parameter:

[ChanNum](#) The channel number established by the DDEInitiate command.

Remarks:

Once a DDE conversation has served its purpose and is no longer needed, your PiXCL script **MUST** terminate the conversation through the [DDETerminate](#) command. If you do not, a conversation can be left open, and Windows will eventually need to be rebooted to clear it.

You should use one DDETerminate command for each conversation you've started.

See the other DDE commands for examples.

Related Commands:

[DDEInitiate](#), [DDEPoke](#) and [DDERequest](#), [DDEExecute](#)

DebugMsgBox

To display the value of any variable in a messagebox, use the [DebugMsgBox](#) command. The current application title is used for the debug message box. For example, if the the title is [Test Application](#), the message box title becomes [Test Application : Debug Message](#).

Syntax: [DebugMsgBox\(String\\$ | Number | FPNumber&\)](#)

Parameter:

[String\\$ | Number | FPNumber&](#) String, integer and floating point static values and variables are automatically converted to a string for display. Floating point numbers will display a maximum of eight decimal places.

Related Commands:

[MessageBox](#)

DeleteColorSpace

PIXCL 5 Command: A previously created logical colour space should be deleted with this command.

Syntax: `DeleteColorSpace(CSHandle, Result)`

Parameters:

CSHandle A handle returned from the `CreateColorSpace` command.

Result 1 if the colour space was deleted, otherwise 0.

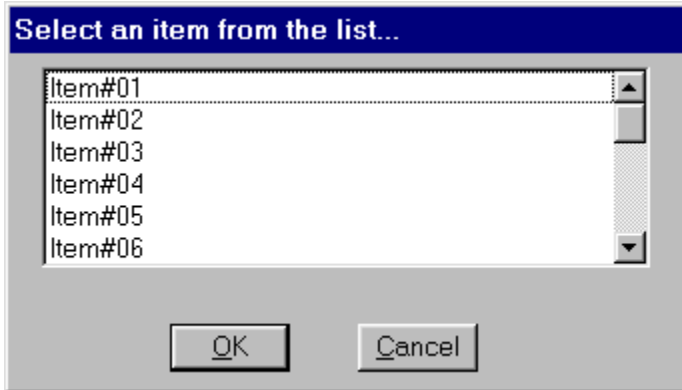
Related Command:

[CreateColorSpace](#)

DialogBox

In PiXCL 4.20 and later, you can define your own custom popup modal dialog boxes, with combinations of push buttons, radio buttons, checkboxes, group boxes, edit controls, static text, List box, and combo boxes, with the `DialogBox` command. These same input controls can be created in the PiXCL client area using other commands.

`DialogBox` is a variable argument list command, with syntax as described below, and is the most complex command in the PiXCL language. Dialog boxes must also appear the same regardless of the current screen resolution and the small or large system font in use. Consequently, the coordinates for the dialog box position, and the controls within the dialog are expressed in what Windows calls **dialog units**, rather than in the pixel coordinates used in most other PiXCL commands. An example custom dialog box appears below.



Syntax: `DialogBox(x1,y1,x2,y2,Title$,STYLE_token, ix,iy,iXsize,iySize,ICON_token, bx1,by1,bx2,by2,BtnLabel$, Controls, ... control definitions)`

Parameters:

| | |
|------------------------------|---|
| <code>x1,y1,x2,y2</code> | The PiXCL client dialog unit coordinates where the custom popup modal dialog is to appear. Coordinates can be negative (e.g. when the PiXCL client is below the dialog), so long as $x1 < x2$ and $y1 < y2$. |
| <code>Title\$</code> | The Title of the custom dialog. |
| <code>STYLE_token</code> | The appearance of the dialog box. <code>CAP_CTR</code> show a title bar, centralise dialog in screen work area. <code>CAP_NCTR</code> show a title bar, dialog relative to client area coords. |
| <code>ix,iy</code> | TL corner position in pixels to draw a built-in icon. |
| <code>ixSize,iySize</code> | Icon size in pixels to draw a built-in icon. |
| <code>ICON_token</code> | The icon token, same as <code>MessageBox</code> command. <code>INFORMATION</code> , <code>EXCLAMATION</code> , <code>QUESTION</code> , <code>STOP</code> , <code>WINLOGO</code> , <code>NOICON</code> , <code>ICON01</code> - <code>ICON19</code> . |
| <code>bx1,by1,bx2,by2</code> | The dialog client coordinates where the mandatory button is to appear. |
| <code>BtnLabel\$</code> | The text label of the mandatory button. |
| <code>NumControls</code> | The number of additional controls that appear in the custom dialog. The current maximum number is 60. See Remarks below for more information. |
| <code>Result</code> | The value returned by the dialog. A dialog always has one button that terminates the dialog. <code>Result</code> is the ordinal number (starting from 1) in which the buttons were defined in the <code>DialogBox</code> command. Please note that regardless of the button pressed, <code>DialogBox</code> will return the string or numeric variables from other control items, if any. |

Control Definitions: Zero to sixty of these must follow in the form...
... `cx1,cy1,cx2,cy2, CONTROL_type, CONTROL_style,Text$, [return_variable,] ...`

| | |
|--|---|
| <code>cx1,cy1,cx2,cy2</code> | dialog unit coordinates for control position. |
| <code>CONTROL_type, CONTROL_style</code> | |
| <code>BTN</code> | Same styles as in <code>Button</code> command. Available styles are ... |
| <code>PUSH</code> | Pushbutton |
| <code>GROUP</code> | Group box. If the vertical size of the group box is 5 dialog units, a single horizontal |

| | |
|---------------------------|---|
| | line is drawn. If the horizontal size of the group box is 1 dialog unit, a single vertical line is drawn. You should set <code>Text\$</code> to a null string in these cases. |
| <code>Text\$</code> | The text string to be displayed within the control. This can be a null string if desired. |
| | The next two styles have an additional argument for the returned state. |
| <code>AUTORADIO</code> | Toggling radio button. |
| <code>AUTOCHECK</code> | Standard toggling check box. |
| <code>Text\$</code> | The text string to be displayed within the control. |
| <code>State</code> | Integer variable of the radio or check state when the dialog is initialized, and when it exits. For pressed or checked, <code>State</code> is set to or returns 1, otherwise it is set to or returns 0. It is possible to initialize all radio buttons to pressed, but when a button is pressed, the states will correct themselves. If you don't initialize the <code>State</code> variable before using it in a <code>DialogBox</code> command, the default is 0. The command automatically sets any non-zero initialize <code>State</code> value to 1. |
| <code>EDIT</code> | Edit control with auto scroll |
| <code>SCREDIT</code> | Edit control with vertical scroll bar |
| <code>STRING</code> | Single line string entry. Carriage return terminates the dialog. |
| <code>MLSTRING</code> | Multi-line string entry. Text wraps around the end of each line automatically, and scrolls vertically as needed. Carriage returns are accepted as new line characters. The maximum size of a string for the edit control is 64KB. |
| <code>NUMBER</code> | positive number entry. For negative numbers, use <code>STRING</code> . |
| <code>PASSWORD</code> | All typed characters are presented as a row of <code>*****</code> . |
| <code>InitText\$</code> | Initializing text variable. This can be a null string. |
| <code>ReturnText\$</code> | Return text variable. For the <code>NUMBER</code> style, this is the ascii string for the number that will need to be converted to an integer variable if required. For the <code>PASSWORD</code> type, this is the actual text typed into the control. |
| <code>STATIC</code> | Static text region. This is commonly presented enclosed in a group box. |
| <code>LEFT</code> | Align text to the left margin |
| <code>CENTER</code> | Align text to the center of the region. |
| <code>Text\$</code> | The text string to be displayed within the static text control. |
| <code>LIST</code> | Single or mult column list box. |
| <code>SINGLE</code> | Displays the list in a single column, with a vertical scroll bar if necessary. |
| <code>MULTI</code> | Displays the multi-column list with a vertical scroll bar if necessary. |
| <code>List\$</code> | The delimited text string list to be displayed within the list control. The delimiter character must be " ", and the last element must have a delimiter appended. |
| <code>Selection\$</code> | The first item in the list if no selection has been made, otherwise the current selection. |
| <code>COMBO</code> | Standard combobox. |
| <code>SIMPLE</code> | The list region is always visible. |
| <code>DROPDOWN</code> | The list region appears when the arrow button is pressed, and the edit control can updated manually. If the list region cannot display all the list, it will automatically scroll downward when you click and drag, but no scroll bar appears. |
| <code>DROPDOWNLIST</code> | Similar to <code>DROPDOWN</code> , but the edit control cannot be manually updated. |
| <code>Text\$</code> | The text string to be displayed within the combobox control. The delimiter character must be " ", and the last element must have a delimiter appended. |
| <code>ReturnText\$</code> | Return text variable that contains the text in the edit control part of the combobox. This can be a null string if no selection has been made.. |

Remarks:

The minimum command is

```
DialogBox (x1,y1,x2,y2,Title$,STYLE_token,ix,iy,ix,isy,ICON01,
bx1,by1,bx2,by2,BtnLabel$,0,Result)
```

which produces a dialog box with a titlebar and one button that when clicked, terminates the dialog box. i.e. not a very useful

dialog.

A common programming error is coding an incorrect value for *NumControls* which will result in either

- not all the controls appearing if *NumControls* is less than the number of controls defined in the command; or
- the dialog box not appearing at all if *NumControls* is greater than the number of controls defined in the command.

Examples:

Please see the **DlgBoxes.pxl** program in the **..learning** subdirectory for some examples of the **DialogBox** command.

You can also create popup dialogs that track when the right mouse is clicked in the client area, and appear with the top left corner near the point on which the mouse was clicked. The **PixelsToDlgUnits** command is needed here.

Related Commands:

[Button](#) [ComboBox](#) [DrawIcon](#) [SetEditControl](#) [GetDialogUnits](#) [DlgUnitsToPixels](#) [PixelsToDlgUnits](#)

DirChange

Sets the current directory to the specified drive and path.

Syntax: [DirChange](#)(*DirectoryName\$,Result*)

Parameters:

[DirectoryName\\$](#) The name of the directory you want to make the current directory. If the directory is on another drive, precede the directory name with the appropriate drive designation.

[Result](#) An integer variable that indicates the outcome of the directory change operation. If the operation was successful, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0.

Remarks:

The directory change operation will fail and PiXCL will assign a value of 0 to [Result](#) if any element of the pathname does not exist.

The current directory is made up of two parts: a disk designator (either a drive letter followed by a colon or a server name and sharename: "\\servername\sharename") and a directory on that disk designator.

You can use a relative path for [DirectoryName\\$](#) (for example, "..") or a fully qualified path.

By using a drive letter followed by a colon for [DirectoryName\\$](#) (for example, "B:"), you can switch to the current directory on the named drive--in effect, making [DirChange](#) equivalent to PiXCL's [DiskChange](#) command.

The presence of "\" at the end of [DirectoryName\\$](#) does not cause the function to fail.

Examples:

This example sets the current directory to the Windows NT SYSTEM32 directory. If the operation fails, the variable Changed is assigned a value of 0, and the program beeps and ends. If the directory is successfully changed, the program checks for the presence of SHELL.DLL. If it is found, a message to that effect is displayed.

```
DirGetSystem(SystemDir$)
DirChange(SystemDir$,Changed)
If Changed=0 Then Goto Beep_End {Dir not changed}
FileExist("SHELL.DLL",Found) {Check for SHELL.DLL}
If Found=0 Then Goto Beep_End
MessageBox(OK,1,INFORMATION,
"SHELL.DLL was found","",Temp)
End
```

Beep_End:

```
Beep
End
```

This next example changes the current directory to C:\LETTERS. It then launches a copy of the command interpreter CMD.EXE:

```
DirChange("C:\LETTERS",There)
If There=1 Then Run("CMD.EXE")
```

The advantage of using the [DirChange](#) command in this example is that C:\LETTERS will be the current directory when the command prompt appears.

Related Commands:

[DirMake](#) [DirRemove](#) [DiskChange](#)

DirExplore

You can select a directory and invoke an Explorer window to show the contents of the directory with the [DirExplore](#) command. This uses the Windows shell and accesses Explorer using a DDE exchange. For more advanced actions with Explorer, you will need to use the DDE commands. The Explorer window that appears will not close automatically: you will have to do this manually or with code.

Syntax: [DirExplore](#)(*DirName*,\$,*Result*)

Parameters:

[DirName](#)\$ The directory that you wish to explore. If the directory does not exist, or is a NULL string, this command has no effect.

[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[DDEInitiate](#) [DDEPoke](#) [DDERequest](#) [DDEExecute](#) [DDETerminate](#)

DirGet

Gets the current directory.

Syntax: [DirGet\(DirectoryName\\$\)](#)

Parameter:

[DirectoryName\\$](#) A string variable that will contain the current directory.

Remarks:

PiXCL returns a fully qualified pathname, including the drive identifier and leading “\” where necessary. Root directories and subdirectories are treated the same. Hence, you might get a return directory **c:\sample** for a subdirectory, and **c:** for a root directory. Note that in both cases there is no trailing “\”. See the second example below.

The current directory is made up of two parts: a disk designator (either a drive letter followed by a colon or a server name and sharename: “\servername\sharename”) and a directory on that disk designator.

Example:

This example gets the current directory and draws it at point (10,10) in the PiXCL window. Next, it makes the Excel directory the current directory, and then shows that it has made the change by drawing the new current directory at (10,20).

```
DirGet (DirectoryName$)
DrawText (10,10,DirectoryName$)
DirChange ("C:\EXCEL",Result)
DirGet (DirectoryName$)
DrawText (10,20,DirectoryName$)
WaitInput ()
```

Another example is where you want to make an application run from any installation directory, including a root directory if necessary.

```
DirGet (SourceDir$)
Image1$ = SourceDir$ + “\image1.bmp”
Image2$ = SourceDir$ + “\image2.bmp”
Image3$ = SourceDir$ + “\image3.bmp”
```

Related Commands:

[DirChange](#) [DirGetWindows](#) [DirGetSystem](#)

DirGetColor

PIXCL 5 command. Gets the Windows 98 / 2000 Color directory where colour profiles are stored..

Syntax: [DirGetColor\(DirectoryName\\$\)](#)

Parameter:

[DirectoryName\\$](#) A string variable that will contain the Windows COLOR directory. In Windows 98, this is usually **c:\windows\SYSTEM\COLOR**. The directories are set to upper case by the Windows API call.

Remarks:

PIXCL returns a fully qualified pathname, including the drive identifier and leading \.

Related Commands:

[InstallColorProfile](#) [UninstallColorProfile](#)

DirGetSystem

Gets the Windows 95/98 SYSTEM or Windows NT / 2000 SYSTEM32 directory.

Syntax: [DirGetSystem\(DirectoryName\\$\)](#)

Parameter:

[DirectoryName\\$](#) A string variable that will contain the Windows NT/2000 SYSTEM32 or Windows 95 SYSTEM directory.

Remarks:

PIXCL returns a fully qualified pathname, including the drive identifier and leading \.

Example:

This example loads the SETUP.INF (located in the Windows SYSTEM directory) into Notepad so that you can view its contents.

```
DirGetSystem(SystemDir$)
SetupInf$ = SystemDir$ + "\setup.inf"
CmdLine$ = "Notepad " + SetupInf$
Run(CmdLine$)
```

Related Commands:

[DirChange](#) [DirGet](#) [DirGetWindows](#) [DirGetTwain](#)

DirGetTwain

Gets the Windows 32 bit TWAIN directory. This is usually **C:\Windows\Twain_32**. Drivers for TWAIN compatible scanners, digital cameras, video recorders and similar devices are stored in this directory, or in subdirectories.

Syntax: [DirGetTwain\(TwainDirectoryName\\$\)](#)

Parameter:

[TwainDirectoryName\\$](#) A string variable that will contain the Windows NT/2000 or Windows 95/98 Twain_32 directory.

Remarks:

PiXCL returns a fully qualified pathname, including the drive identifier and leading \.

Related Commands:

[DirChange](#) [DirGet](#) [DirGetWindows](#)

DirGetWindows

Gets the Windows directory.

Syntax: [DirGetWindows](#)(*DirectoryName*\$)

Parameter:

[DirectoryName](#)\$ A string variable that will contain the Windows directory.

Remark:

PiXCL returns a fully qualified pathname, including the drive identifier and leading \.

Example:

This example tests for the presence of TARTAN.BMP in the Windows directory. If it's found, the program loads it into Paintbrush for editing. Otherwise, an error message is displayed.

```
DirGetWindows (WindowsDir$)
Tartan$ = WindowsDir$ + "\tartan.bmp"
FileExist(Tartan$,Exist)
CommandLine$ = "pbrush " + Tartan$
If Exist = 1 Then Run(CommandLine$) | End
MessageBox(OK,1,INFORMATION,
    "Couldn't find tutor","Not found",Temp)
```

Related Commands:

[DirChange](#) [DirGet](#) [DirGetSystem](#) [DirGetTwain](#)

DirListFiles

List the set of the filenames located in a specific directory.

Syntax: [DirListFiles\(Path\\$,Delimiter\\$,Number,List\\$\)](#)

Parameters:

| | |
|-----------------------------|--|
| Path\$ | The disk, directory and file type to be listed. To list all files, use *.*. To list all files of particular type, use e.g. *. pxl . |
| Delimiter\$ | The delimiter character e.g. “ ”. |
| Number | The number of files of the type in Path\$ in the directory. |
| List\$ | The set of files found in the directory. |

Related Commands:

[FileExist](#) [DirMake](#) [DirRemove](#)

DirMake

Creates a new directory.

Syntax: [DirMake](#)(*DirectoryName*,\$, *Result*)

Parameters:

- [DirectoryName](#)\$ The name of the directory you want to create. If the directory is on another drive, precede the directory name with the appropriate drive designation.
- [Result](#) An integer variable that indicates whether the directory was successfully created. If it was, this variable is assigned a value of 1. If it was not successfully created. For example, an element of the pathname does not exist: this variable is assigned a value of 0.

Remarks:

The directory will not be created and PiXCL will assign [Result](#) a value of 0 under the following circumstances:

- Any element of the pathname does not exist.
- A directory with the same name already exists.

[DirectoryName](#)\$ may be a relative path (for example, "..\temp") or a fully qualified path.

If you want to determine whether a directory already exists before trying to create it, use the [FileExist](#) command.

Example:

The following example creates a \TXTFILES subdirectory off of C:\WINDOWS. If the operation fails--for example, the directory already exists--the Created and False variables are equal, and the program beeps and ends. If the operation is a success, the program displays a message to that effect.

```
False = 0
DirMake("C:\WINDOWS\TXTFILES",Created)
If Created=False Then Beep | End
MessageBox(OK,1,INFORMATION,"\TXTFILES successfully created",
           "",Temp)
```

Related Commands:

[DirChange](#) [DirRemove](#) [DiskChange](#) [FileExist](#)

DirMakePath

PIXCL 5 command. Creates a new directory path.

Syntax: [DirMakePath](#)(*DirectoryPath*,\$, *Result*)

Parameters:

[DirectoryPath](#)\$ The name of the directory path you want to create. If the directory is on another drive, precede the directory name with the appropriate drive designation.

[Result](#) An integer variable that indicates whether the directory path was successfully created. If it was, this variable is assigned a value of 1. If it was not successfully created. For example, an element of the pathname does not exist: this variable is assigned a value of 0.

Remarks:

The path to be created is terminated by the last trailing backslash. For example, a path "[C:\Test1\Test2\Test3.dir\](#)" creates "[C:\Test1\Test2\Test3.dir\](#)" while "[C:\Test1\Test2\Test3.dir](#)" creates "[C:\Test1\Test2](#)".

If you want to determine whether a directory already exists before trying to create it, use the [FileExist](#) command. See also the set of Path* commands.

Related Commands:

[DirMake](#)

DirRemove

Deletes an existing directory.

Syntax: *DirRemove*(*DirectoryName*\$,*Result*)

Parameters:

DirectoryName\$ The name of the directory you want to delete. If the directory is on another drive, precede the directory name with the appropriate drive designation.

Result An integer variable that indicates whether deletion of the directory was successful. If the deletion was a success, this variable is assigned a value of 1. If the deletion was not a success—for example, an element of the pathname does not exist -- the *Result* variable is assigned a value of 0.

Remarks:

The deletion will fail and *Result* will be assigned a value of 0 under the following circumstances:

- Any element of the pathname does not exist.
- The specified directory is also the current directory.
- The directory still contains files (or subdirectories).

DirectoryName\$ may be a relative path (for example, "..\temp") or a fully qualified path.

Example:

The following example deletes the C:\WINDOWS\MYFILES directory. If the operation fails—for example, the directory doesn't exist—the Deleted and False variables are equal, and the program beeps and ends. If the deletion is successful, the program displays a message to that effect:

```
False=0
DirRemove("C:\WINDOWS\MYFILES",Deleted)
If Deleted=False Then Beep | End
MessageBox(OK,1,INFORMATION,
    "\WINDOWS\MYFILES successfully deleted",
    "",Temp)
```

Related Commands:

[DirChange](#) [DirMake](#) [DiskChange](#)

DiskChange

Makes another disk drive the active drive.

Syntax: `DiskChange(DriveLetter$,Result)`

Parameters:

DriveLetter\$ A string indicating the letter of the drive you want to make the active drive.

Result An integer variable that indicates whether the disk change was successful. If it was successful, this variable is assigned a value of 1. If it was not successful, this variable is assigned a value of 0.

Remarks:

When you change disk drives with this command, the current directory settings are not affected.

A colon is optional after the driver letter. In fact, any characters after the drive letter are ignored.

Example:

This program makes drive E the current drive. If the disk change fails--for example, the drive doesn't exist--the program beeps and ends. If the disk change is successful, the program displays a message indicating that E is now the active drive.

```
False = 0
DiskChange("E",Changed)
If Changed=False Then Beep | End
MessageBox(OK,1,INFORMATION,
    "Drive E is the active drive",
    "",Temp)
```

Related Commands:

All the Directory and File commands.

DlgUnitsToPixels

Converts a dialog units coordinate to a client pixel coordinate.

Syntax: `DlgUnitsToPixels(Dx,Dy,Px,Py)`

Parameters:

Dx,Dy The dialog box coordinate.

Px,Py The returned pixel coordinate.

Remarks:

Converting between pixels and dialog coordinates will result in some loss of precision.

Related Commands:

[DialogBox](#) [PixelsToDlgUnits](#)

Double

PIXCL 5.1 command. An integer or int64 number can be converted to a double (64 bit floating point).

Syntax: `Double(Integer|Integer64#, Fp64#&)`

Parameters:

`Integer, Integer64` Any static number or integer variable
`Fp64#&` The resulting double variable.

Related Command:

[Float](#)

DragAcceptFile

With this command, and the related [GetDragList](#) command, you can drag and drop single or multiple files into your PiXCL application for processing by PiXCL code. These files will often be images, but can also be text files, or any filename on which your PiXCL application performs some process.

Syntax: [DragAcceptFile\(ENABLE|DISABLE,label\)](#)

Parameters:

[ENABLE|DISABLE](#) Enables or disables the drag accept file function.

[label](#) The label in your program that handles the file or file list.

Remarks:

Default is [DISABLE](#) if your program does not include a [DragAcceptFile](#) command. If you issue multiple [DragAcceptFile](#) commands, the most recent command jump-to label becomes the current label.

When you try to drag and drop a file into a PiXCL application, the cursor changes to the standard drop enable or drop disable style once the mouse is in the client area. If enabled, once you click the left mouse, the cursor changes back the arrow.

For example, to load a set of image filenames from Explorer, select the set of files, click and hold the left mouse key, drag over to your PiXCL application, and release the mouse. The set of filenames will be then available to your PiXCL application via the [GetDragList](#) command.

Related Commands:

[DropFileServer](#) [GetDragList](#)

DrawAnimatedRects

The [DrawAnimatedRects](#) function draws a wire-frame rectangle and animates it to indicate the opening of an icon or the minimizing or maximizing of a window or region. This is what you see when Windows 95/98/2000 opens and closes application windows, or minimizes them on to the task bar.

Syntax: [DrawAnimatedRects](#)(*WindowName*,\$
fx1,fy1,fx2,fy2, tx1,ty1,tx2,ty2,
OPEN | CLOSE | CAPTION, Result)

Parameters:

| | |
|---------------------------------|--|
| WindowName \$ | The exact Title string. The EnumWindows command is useful to get windownames. If you use a null string "" the PiXCL application window text and icon is drawn. |
| fx1,fy1,fx2,fy2 | The client area co-ordinates of the rectangle from which the rectangle is to be drawn. |
| tx1,ty1,tx2,ty2 | The client area co-ordinates of the rectangle to which the rectangle is to be drawn. |
| OPEN | Use to open or enlarge a window or region. |
| CLOSE | Use to close or compact a window or region. |
| CAPTION | Use to take the title bar and caption of the target window for the animate process. |
| Result | 1 if success, otherwise zero. |

Related Commands:

None

DrawArc

Draws an elliptical arc using the current pen. To draw the arc, you use the points $(x1,y1)$ and $(x2,y2)$ to define a rectangle that bounds the ellipse containing the arc. You then use the parameters $(x3,y3)$ to specify the point on the ellipse where the arc starts, and the parameters $(x4,y4)$ to specify where it ends. Note that when the GDI sweeps the arc, it begins at $(x3,y3)$ and moves in a counterclockwise direction towards $(x4,y4)$.

Syntax: `DrawArc(x1,y1,x2,y2,x3,y3,x4,y4)`

Parameters:

| | |
|--------------------|--|
| <code>x1,y1</code> | The upper-left corner of the rectangle bounding the ellipse containing the arc. |
| <code>x2,y2</code> | The lower-right corner of the rectangle bounding the ellipse containing the arc. |
| <code>x3,y3</code> | The starting point of the arc on the ellipse. |
| <code>x4,y4</code> | The ending point of the arc on the ellipse. |

Remark:

The points $(x3,y3)$ and $(x4,y4)$ do not have to lie precisely on the ellipse. If they do not, however, the GDI uses points on the ellipse that are the shortest distance from $(x3,y3)$ and $(x4,y4)$.

Examples:

This example sets the coordinate system to pixels, then draws an arc within the rectangle defined by the points (30,20) and (200,180). It sweeps the arc starting from the point (200,20) and moving in a counterclockwise direction to the point (30,20).

```
UseCoordinates(PIXEL)
DrawArc(30,20,200,180,200,20,30,20)
WaitInput()
```

The next program draws an arc by asking you to click on each of the four points needed to define the arc. The first point it asks for is the upper-left corner of the rectangle bounding the ellipse containing the arc, and the second is the lower-right corner of that same rectangle. The program then draws a temporary ellipse using the points you've defined so that you can select the third and fourth points needed for the arc--its starting and ending points. After you've clicked on these points on the ellipse, the program clears the screen and draws the arc using the `DrawArc` command. It then loops back up for you to specify another arc.

```
{Set up the environment}
SetWindow(MAXIMIZE)
UseCoordinates(PIXEL)
UseFont("Terminal",10,10,
        NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)

Arc1:
UsePen(DOT,1,0,0,0) {Dotted pen for the temp ellipse}
SetMouse(0,0,700,600,Arc2,x1,y1)
Text1$ = "Click on upper-left corner of rectangle bounding arc"
DrawText(10,300,Text1$)
Goto Get_Input

Arc2:
SetMouse(0,0,700,600,Arc3,x2,y2)
Text2$ = "Click on lower-right corner of rectangle bounding arc"
DrawText(10,300,Text2$)
Goto Get_Input

Arc3:
DrawEllipse(x1,y1,x2,y2)           {Draw a temporary ellipse}
```



```

SetMouse(0,0,700,600,Arc4,x3,y3)
Text3$ = "Click on arc's starting point"
Len(Text2$,Length2)
Len(Text3$,Length3)
Difference = Length2 - Length3
Pad(Text3$,Difference)           {Pad text with spaces}
DrawText(10,300,Text3$)
Goto Get_Input

Arc4:
SetMouse(0,0,700,600,Arc_End,x4,y4)
Text4$ = "Click on arc's ending point"
Len(Text4$,Length4)
Difference = Length2 - Length4
Pad(Text4$,Difference)           {Pad text with spaces}
DrawText(10,300,Text4$)
Goto Get_Input

Arc_End:
DrawBackground                   {Clear the temporary ellipse}
UsePen(SOLID,1,0,0,0)            {Use a solid black pen}
DrawArc(x1,y1,x2,y2,x3,y3,x4,y4)
Goto Arc1

Get_Input:
WaitInput()

```

Related Commands:

[DrawEllipse](#) [UsePen](#) [UseBrush](#)

DrawBackground

Draws a window's background using the current background color. If the window has any contents, they are overwritten in the process.

Syntax: [DrawBackground](#)

Remark:

You control the current background color using the UseBackground command. (The current background mode, [TRANSPARENT](#) or [OPAQUE](#), has no effect on the DrawBackground command's behavior.)

Example:

This example sets the background color to green using the UseBackground command and then draws the background. Next, it draws some text in the window, waits for 2 seconds, and erases the text by issuing the DrawBackground command again.

```
UseBackground(TRANSPARENT, 0, 255, 0)
DrawBackground
DrawText(10, 10, "The effect of text")
WaitInput(2000)      {Pause for 2 seconds}
DrawBackground      {Erase the text}
WaitInput()
```

Related Commands:

[GetBackground](#) [UseBackground](#)

DrawBackgroundRegion

PiXCL has a foreground and background copy of what you see in the PiXCL application client area on your screen. It is possible to draw in either the foreground or background separately, or both, which is the default setting when the PiXCL application starts.

[DrawBackgroundRegion](#) is designed to be used with simple animation, where you want to copy a region of the client area to the foreground. For example, if an animation bitmap has a cell size of 23x64 pixels, you can specify just the background region under the cell to be refreshed. If you use the [SetDrawMode](#) command which works with all the [DrawBitmap](#) commands, you can use the background memory image for double buffering animation.

Syntax: [DrawBackgroundRegion\(x1,y1,x2,y2\)](#)

Parameters:

[x1,y1](#) The top left corner of the region to be redrawn.
[X2,y2](#) The bottom right corner of the region to be drawn.

Remarks:

This command is NOT the same as the [DrawBackground](#) command, which sends a message to PiXCL to redraw the entire screen image with the current background color. Rather, it takes a copy of the specified region and passes it to the foreground. [DrawBackgroundRegion](#) is NOT affected by the [SetDrawMode](#) command.

Related Commands:

[DrawBackground](#) [SetDrawMode](#) [SetROPcode](#) [DrawBitmap](#)

DrawBitmap, DrawPreviewBitmap

Places the contents of a bitmap (BMP, JIF, JPEG, PCD, PCX, PNG, PPM, PSD, RAS, RLE, TGA, TIF) file at a specified location on the screen. The bitmap can be a Preview size which has maximum dimensions of 256x256, with the same aspect ratio as the full image.

Syntax: `DrawBitmap(x,y,Filename$)`
`DrawPreviewBitmap(x,y,Filename$)`

Parameters:

| | |
|-------------------------|---|
| <code>x</code> | The x-coordinate of the upper-left corner of the bitmap. |
| <code>y</code> | The y-coordinate of the upper-left corner of the bitmap. |
| <code>Filename\$</code> | The name of the bitmap file. Include the path if the bitmap file is not in the current directory. |

Remarks:

`DrawBitmap` supports 256-color and 24-bit bitmaps. Be aware, however, that this type of bitmap can consume a substantial amount of memory. For example, a bitmap containing an entire screen image of a 1024x768-pixel 256-color SuperVGA is over 768K.

When you use the `DrawBitmap` command to place a bitmap in an PiXCL window, PiXCL stores the bitmap image in Windows global memory, as well as in the application memory and display contexts. These contexts are, in effect, the actual PiXCL window as you see it, one stored somewhere in global memory (the memory context), and the other on your video card memory (the display context). When your script ends, PiXCL removes the bitmap(s) from memory and recovers the space it occupies. The image you see drawn in the client area is a copy of the bitmap stored elsewhere in memory.

In most cases, we advise you to remove a bitmap from memory once you have loaded the image file. In most cases, this will not have an undesirable effects. What this approach does is free up the image file global memory, while leaving the memory and display contexts alone.

If you are retrieving the same bitmap from disk and the disk bitmap has changed from the bitmap in program memory, you MUST use `FreeBitMap` before you reload with `DrawBitmap`, or else the original bitmap in the memory context will be displayed.

One way to reduce a bitmap's memory consumption is to convert it to RLE (Run Length Encoded) bitmap format. (JASC Paint Shop Pro, a popular Windows shareware program, is handy for doing so.) By converting a bitmap to RLE format, you can usually compress it to half its original size, sometimes smaller, with no ill-effects. Like many Windows programs, PiXCL fully supports RLE bitmap files. You could also use the format conversion ability within PiXCL, and use the `SaveBitmap` command to save an image into RLE format.

If you draw one 256-color bitmap after another in the same PiXCL window, the first bitmap will likely change color as the second bitmap's color palette is realized; this occurs instantly, just before the second bitmap is appears on the screen (the second bitmap appears normally). The `SetColorPalette` command was introduced to try to mitigate this problem. By using `SetColorPalette(GENERATE)`, you can have PiXCL generate an evenly distributed color palette that is used by both bitmaps. Neither bitmap will appear optimal, but one bitmap's colors will not override the other's. If this is a problem for you, we suggest that you install video card with 2MB or more video memory, as this greatly increases the available palette colors.

Example:

This example reads the arches bitmap file (ARCHES.BMP) in the Windows directory and draws its contents beginning in the upper-left corner of the window.

```
DirGetWindows (WinDir$)
Bitmap$=WinDir$+"\ARCHES.BMP"
DrawBitmap(0,0,Bitmap$)
WaitInput()
```

This next example places two bitmaps on the screen: the chitz bitmap (CHITZ.BMP) starting at (5,5), and the ball bitmap

(BALL.BMP) starting at (100,5):

```
DirGetWindows (WinDir$)
Bitmap1$=WinDir$+"\ARCHES.BMP"
DrawBitmap (5, 5, Bitmap1$)
Bitmap2$=WinDir$+"\BALL.BMP"
DrawBitmap (100, 5, Bitmap2$)
WaitInput ()
```

Remarks:

You can draw into the foreground or background by using the [SetDrawMode](#) command.

Related Commands:

[DrawSizedBitmap](#) [FreeBitmap](#) [FreeBitmapAll](#) [GetBitMapDim](#) [SetColorPalette](#) [DrawZoomedBitmap](#) [LoadBitmap](#)

DrawBitMapExt

This command offers some additional bitmap draw options according to the TOKEN used. It places the contents of a bitmap file at a specified location on the screen.

Syntax: [DrawBitmapExt\(x,y,Filename\\$,TOKEN\)](#)

Parameters:

| | |
|-------------------|---|
| <i>x</i> | The x-coordinate of the upper-left corner of the bitmap. |
| <i>y</i> | The y-coordinate of the upper-left corner of the bitmap. |
| <i>Filename\$</i> | The name of the bitmap file. Include the path if the bitmap file is not in the current directory. |

[BLACKONWHITE](#)

[COLORONCOLOR](#)

[HALFTONE](#)

[WHITEONBLACK](#)

Remarks:

These commands will often appear to have no or little effect. See what happens if you use the [SetROPcode](#) command. You can draw into the foreground or background by using the [SetDrawMode](#) command.

Related Commands:

[DrawSizedBitmap](#) [FreeBitmap](#) [FreeBitmapAll](#) [GetBitMapDim](#) [SetColorPalette](#) [DrawZoomedBitmap](#) [LoadBitmap](#)

DrawBitmapPoint

This command is provided when you need to identify a point in a bitmap window, such as when selecting with the [SetBMWMouse](#) and [SetBMWRightMouse](#) commands. The point is written in the current pen color for 24 bit images, and for 8 bit paletted images, the current pen color if it exists in the palette, otherwise as pixel value 255.

| | |
|---------------------|---|
| Point style CROSS | × |
| Point style PLUS | + |
| Point style CIRCLE | ○ |
| Point style BOX | □ |
| Point style DIAMOND | ◇ |
| Point style DOT | · |

Syntax: [DrawBitmapPoint](#)(*WindowID*,*Filename*,\$*x*,*y*,*style_TOKEN*,*Result*)

Parameters:

| | |
|---------------------------------------|---|
| WindowID | The bitmap window ID returned by a DrawBitmapWindow command. |
| Filename \$ | The image that is displayed in the bitmap window. This image must be in the PIXCL image list. |
| x , y | The bitmap coordinates for the point that is to be written. Note that this is NOT the client area coordinate. |
| CROSS | an X, as shown above. |
| PLUS | a + , as shown above. |
| CIRCLE | a circle with a center dot, as shown above. |
| BOX | a box with a center dot, as shown above. |
| DIAMOND | a diamond with a center dot, as shown above. |
| DOT | a 3x3 dot, as shown above. |
| Result | 1 of the operation succeeds, otherwise 0. |

Remarks:

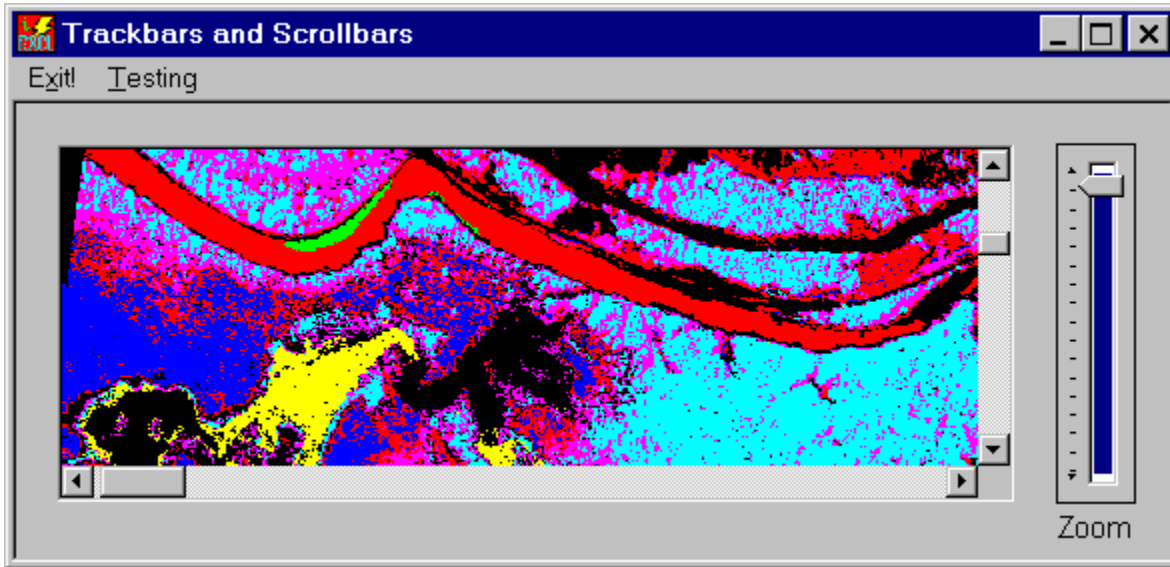
[DrawBitmapPoint](#) writes data to the bitmap loaded into memory. You can save the image, and the points will appear in the new image. If the points are written on a temporary basis, prior to some other operation, you should reload the image by issuing a [FreeBitmap](#) and a [LoadBitmap](#) or [DrawBitmap](#) command sequence.

Related Commands:

[DrawBitmapWindow](#) [DrawPoint](#) [SetBMWMouse](#) [UsePen](#)

DrawBitmapWindow

This command is provided when you want to see a large image at full resolution, but in a limited region of the PiXCL application client area. You can specify the position and dimensions of the display window, and it appears with both vertical and horizontal scroll bars, unless the size of the window exactly matches the dimensions of the image to be displayed. The image that is loaded into the bitmap window is also written to the PiXCL bitmap list and can be listed with the [ListLoadedBitmap](#) command, and removed with the [FreeBitmap](#) command.



Up to eight bitmap windows can be displayed. Each window can be independently scrolled with the scrollbars, and zoomed with the [ZoomBitmapWindow](#) command, perhaps controlled by a trackbar as in the image above. The default bitmap window title is the name of the image displayed there: You can change this with the [BMWinTitle](#) command.

Syntax: [DrawBitmapWindow](#)(*x1,y1,x2,y2*, *Filename\$*, *STYLE_token*, *WindowID*)

Parameters:

x1,y1,x2,y2

The PiXCL application **client** area coordinates for the bitmap window. If a sizing or captioned window is defined, it can be moved around the PiXCL client area with the mouse.

Filename\$

The name of the image to be displayed. This is the default window title. If the image is not already loaded into the PiXCL image list, it is loaded automatically at full resolution. If you want to display preview images in a bitmap window, use the [LoadBitmap](#)(*Imagename\$,PREVIEW*) or the [DrawPreviewBitmap](#) commands.

STYLE_token

[NOCAPTION_NOSIZE](#): create a non-sizing window without a caption.
[NOCAPTION_SIZE](#): create a sizing window without a caption.
[CAPTION_NOSIZE](#): create a non-sizing window with a caption.
[CAPTION_SIZE](#): create a sizing window with a caption.

WindowID

The ID of the window created. This variable is usually needed to be used with the [CloseBitmapWindow](#) command, and also for the [FlashBMWindow](#) and [SetBMWMouse](#) commands. If *Filename\$* cannot be located or loaded, the operation fails and *WindowID* returns 0.

Remarks:

The bitmap window caption, if enabled, is the image *Filename\$*. Depending on the image dimensions, the size of the slider buttons will vary. This is normal Windows operation.

Image processing commands do not directly display the results. To display a result image, use the [ZoomBitmapWindow](#) in [INCREMENT](#) mode, with value set to 0.

You have left and right mouse access to a bitmap window with the [SetBMWMouse](#) and [SetBMWRightMouse](#) commands, which are very similar in operation to the [SetMouse](#) commands for the PiXCL client area.

A maximum of eight bitmap windows can be defined. PiXCL keeps track of any available window entries, so it is possible to delete one window and add another. If eight windows are already defined, no additional bitmap windows will be created. If the bitmap window has a titlebar enabled, the window can be closed by clicking the close button, which frees the relevant bitmap record for further use.

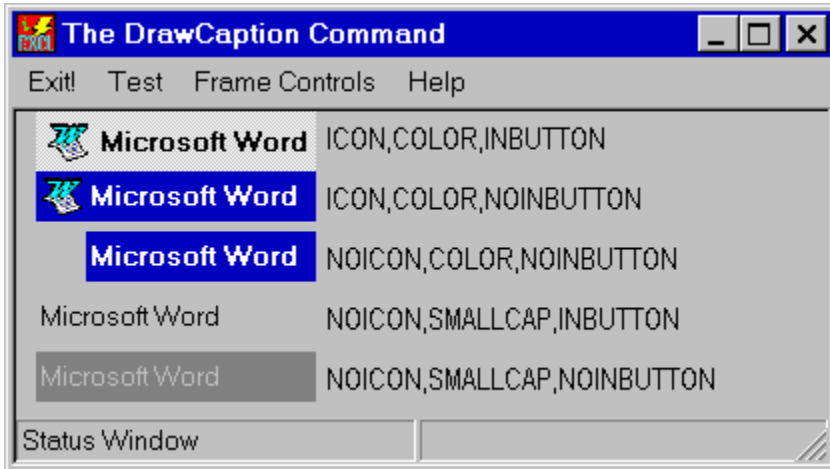
Related Commands:

[BMWinTitle](#) [ChangeBmwImage](#) [CloseBitmapWindow](#) [FlashBMWindow](#) [GetBMWZoom](#) [SetBMWMouse](#)
[SetBMWRightMouse](#) [TileBitmapWindows](#) [ZoomBitmapWindow](#)

DrawCaption

Any window caption and its icon can be drawn in the client area, even if the window is not visible. The background of the rectangle is the same as the current title bar background system color. The DrawCaption command is used to draw application buttons similar to those in the Windows task bar.

Syntax: `DrawCaption(WindowName$,x1,y1,x2,y2,
(NO)ICON, COLOR|SMALLCAP,
(NO)INBUTTON, Result)`



Parameters:

- WindowName\$** The exact Title string. The [EnumWindows](#) command is useful to get windownames. If you use a null string " " the PiXCL application window text and icon is drawn.
- x1,y1,x2,y2** The client area co-ordinates of the rectangle in which the information is to be drawn.
- NOICON | ICON** Token that displays or hides the default icon for the chosen window.
- COLOR** Use system colors for background and text.
SMALLCAP Use gray background and white text.
- (NO)INBUTTON** Set a light gray or dark gray background to the specified drawing rectangle.
- Result** Zero if the target window cannot be located, otherwise 1.

Remarks:

If the window title text is longer than the defined rectangle, the text is truncated, the same as in the Windows task bar.

Related Commands:

[EnumWindows](#)

DrawChord

Draws a chord—a region bounded by the intersection of a line and an ellipse. To draw a chord, you use the points $(x1,y1)$ and $(x2,y2)$ to define a rectangle that bounds the ellipse that is part of the chord. You then specify the line that intersects the ellipse using the points $(x3,y3)$ and $(x4,y4)$. The command draws the chord's border using the current pen and fills its interior using the current brush.

Syntax: `DrawChord(x1,y1,x2,y2,x3,y3,x4,y4)`

Parameters:

| | |
|--------------------|---|
| <code>x1,y1</code> | The upper-left corner of the rectangle bounding the ellipse that is part of the chord. |
| <code>x2,y2</code> | The lower-right corner of the rectangle bounding the ellipse that is part of the chord. |
| <code>x3,y3</code> | A point on the line that intersects the ellipse. |
| <code>x4,y4</code> | A second point on the line that intersects the ellipse. |

Remark:

The points $(x3,y3)$ and $(x4,y4)$ do not have to lie precisely on the ellipse. If they do not, however, the GDI uses points on the ellipse that are the shortest distance from $(x3,y3)$ and $(x4,y4)$.

Examples:

This example draws a chord by using the points (40,30) and (210,190) to define the rectangle that bounds the ellipse that is part of the chord. The points used for the line that intersects the ellipse are (210,30) and (40,30). The chord is drawn with the default black pen and a solid aqua brush.

```
UseCoordinates(PIXEL)
UseBrush(SOLID,0,255,255)
DrawChord(40,30,210,190,210,30,40,30)
WaitInput()
```

This next example is a variation of one shown for the `DrawArc` command. It draws a chord by prompting you to click on the four points needed to define a chord. The first point is the upper-left corner of the rectangle bounding the ellipse that is part of the chord; the second is the lower-right corner of that same rectangle; the third is a point on the line bounding the chord; and the fourth is another point on that same line. After you've clicked on all four points, the program draws the chord using the `DrawChord` command, then loops back up for you to specify another chord.

```
{Set up the environment}
SetWindow(MAXIMIZE)
UseCoordinates(PIXEL)
UseFont("Terminal",10,10,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
Chord1:
UsePen(DOT,1,0,0,0)      {Use dotted pen for temporary ellipse}
UseBrush(NULL,0,0,0)    {Use hollow brush for temporary ellipse}
SetMouse(0,0,65000,65000,Chord2,x1,y1)
Text1$ =
    "Click on upper-left corner of rectangle bounding chord "
DrawText(10,300,Text1$)
Goto Get_Input
Chord2:
SetMouse(0,0,65000,65000,Chord3,x2,y2)
Text2$ =
    "Click on lower-right corner of rectangle bounding chord"
DrawText(10,300,Text2$)
```

```

Goto Get_Input
Chord3:
  DrawEllipse(x1,y1,x2,y2)  {Draw temporary ellipse}
  SetMouse(0,0,65000,65000,Chord4,x3,y3)
  Text3$ = "Click on one point of line defining chord"
  Len(Text2$,Length2)
  Len(Text3$,Length3)
  Difference = Length2 - Length3
  Pad(Text3$,Difference)      {Pad text with spaces}
  DrawText(10,300,Text3$)
  Goto Get_Input
Chord4:
  SetMouse(0,0,65000,65000,Chord_End,x4,y4)
  Text4$ = "Click on second point of line defining chord"
  Len(Text4$,Length4)
  Difference = Length2 - Length4
  Pad(Text4$,Difference)      {Pad text with spaces}
  DrawText(10,300,Text4$)
  Goto Get_Input

Chord_End:
  DrawBackground              {Clear the temporary ellipse}
  UsePen(SOLID,1,0,0,0)       {Use a solid black pen}
  UseBrush(SOLID,255,0,0)     {Use a solid red brush}
  DrawChord(x1,y1,x2,y2,x3,y3,x4,y4)
  Goto Chord1

Get_Input:
  WaitInput()

```

Related Commands:

[UseBrush](#), [UsePen](#)

DrawCircle

Draws a circle. To specify the circle, you use the centre point *(x1,y1)* the *radius*. The command uses the current pen to draw the border and the current brush to fill the interior.

Syntax: `DrawCircle(x1,y1, radius)`

Parameters:

x1,y1 The centre point of the circle.

radius The radius of the circle.

Related Commands:

[DrawEllipse](#)

DrawEdgeRectangle

The [DrawEdgeRectangle](#) function draws one or more edges of a rectangle, in the current system background color, which is usually set to light gray (192, 192, 192). [DrawEdgeRectangle](#) is useful to create status bars and 3D buttons with multiple states. It is also possible to add one of the icons built into PiXCL as button Bitmap, using the [DrawIcon](#) command.

Syntax: [DrawEdgeRectangle\(x1,y1,x2,y2,TOKEN#1,TOKEN#2,TOKEN#3\)](#)

Parameters:

[x1,y1,x2,y2](#) The client area co-ordinates of the rectangle to be drawn.

TOKEN#1

Specifies the type of inner and outer edge to draw.

| <u>Value</u> | <u>Meaning</u> |
|----------------------------|---|
| BUMPEDGE | A two pixel wide 3D effect like a raised line |
| ETCHEDGE | A two pixel wide 3D effect like a scored or etched line. |
| RAISEEDGE | A 3D effect that makes the rectangle appear raised one pixel. |
| SUNKENEDGE | A 3D effect that makes the rectangle appear sunken one pixel. |

TOKEN#2

Token #2 and token #3 specify the type of border for the rectangle, and are combined when the command is executed.

| <u>Value</u> | <u>Meaning</u> |
|----------------------------------|--|
| ADJUST | Rectangle to be adjusted to leave space for client area. |
| BOTTOM | Bottom of border rectangle. |
| BOTTOMLEFT | Bottom and left side of border rectangle. |
| BOTTOMRIGHT | Bottom and right side of border rectangle. |
| DIAGONAL | Diagonal border. |
| DIAG_ENDBTMLEFT | Diagonal border. The end point is the bottom-left corner of the rectangle; the origin is top-right corner. |
| DIAG_ENDBTMRIGHT | Diagonal border. The end point is the bottom-right corner of the rectangle; the origin is top-left corner. |
| DIAG_ENDTOPLEFT | Diagonal border. The end point is the top-left corner of the rectangle; the origin is bottom-right corner. |
| DIAG_ENDTOPRIGHT | Diagonal border. The end point is the top-right corner of the rectangle; the origin is bottom-left corner. |

TOKEN#3

| <u>Value</u> | <u>Meaning</u> |
|--------------------------|---|
| LEFT | Left side of border rectangle. |
| MIDDLE | Interior of rectangle to be filled. |
| RECT | Entire border rectangle. |
| RIGHT | Right side of border rectangle. |
| TOP | Top of border rectangle. |
| TOPLEFT | Top and left side of border rectangle. |
| TOPRIGHT | Top and right side of border rectangle. |

Related Commands:

DrawRectangle, DrawStatusText , all the SetMouse commands, Button

DrawEllipse

Draws an ellipse (or circle). To specify the ellipse, you use the points $(x1,y1)$ and $(x2,y2)$ to define a rectangle that bounds the ellipse. The command uses the current pen to draw the border and the current brush to fill the interior.

Syntax: `DrawEllipse(x1,y1,x2,y2)`

Parameters:

`x1,y1` The upper-left corner of the rectangle bounding the ellipse.

`x2,y2` The lower-right corner of the rectangle bounding the ellipse.

Examples:

The following example draws an ellipse that is bounded by the rectangle specified by the points (20,30) and (80,60). The ellipse is drawn using a solid black pen (the default) and a solid yellow brush.

```
UseBrush (SOLID,255,255,0)
DrawEllipse(20,30,80,60)
WaitInput()
```

This second example prompts you to click on the points of the rectangle bounding an ellipse. It then draws the ellipse using the default pen and a solid green brush.

```
{Set up the environment}
  SetWindow (MAXIMIZE)
  UseCoordinates (PIXEL)
  UseFont ("Terminal",10,10,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
  UseBrush (SOLID,0,255,0)   {Use a solid green brush}
  GetScreenCaps (HORZRES,MaxX)
  GetScreenCaps (VERTRES,MaxY)

Ellipse1:
  SetMouse (0,0,MaxX,MaxY,Ellipse2,x1,y1)
  DrawText (10,300,
    "Click on upper-left corner of rectangle bounding ellipse ")
  Goto Get_Input
Ellipse2:
  SetMouse (0,0,MaxX,MaxY,Ellipse3,x2,y2)
  DrawText (10,300,
    "Click on lower-right corner of rectangle bounding ellipse")
  Goto Get_Input
Ellipse3:
  DrawEllipse(x1,y1,x2,y2)
  Goto Ellipse1

Get_Input:
  WaitInput()
```

Related Commands:

[DrawCircle](#) [UseBrush](#), [UsePen](#)

DrawFlood

Fills in any enclosed shape or area using the current brush. It begins at the point specified by the *x* and *y* parameters and fills in all directions until it reaches the color boundary specified by the *r, g, b* parameters.

Syntax: `DrawFlood(x,y,r,g,b)`

Parameters:

- x* The x-coordinate of a point inside the area you want to fill.
- y* The y-coordinate of a point inside the area you want to fill.
- r,g,b* Specifies the color of the boundary up to which PIXCL is to fill.

Example:

This example draws a series of shapes on the screen using a blue pen and a hollow brush. It then sets the brush to solid red and asks you to click within an area to flood it with red. When you click on an area, the program branches to `Flood_Mouse`, where the `DrawFlood` command fills in the selected area using the current brush. (Notice that the *r,g,b* parameters are 0,0,255, causing the command to fill an area until it encounters a blue border.) The program stays in a loop, allowing you to flood as many areas as you like.

```
{Set up the mouse, pen, and brush}
SetWindow(MAXIMIZE)
SetMouse(0,0,700,600,Flood_Mouse,Flood_x,Flood_y)
UsePen(SOLID,3,0,0,255) {Use a 3 pixel wide blue pen}
UseBrush(NULL,0,0,0)    {Use hollow brush for random shapes}

{Fill the window with random shapes}
DrawRectangle(5,15,200,150)
DrawRectangle(55,31,82,50)
DrawRectangle(105,41,120,140)
DrawRectangle(50,50,100,150)
DrawRectangle(105,50,120,133)
DrawRectangle(5,120,180,125)
DrawEllipse(30,30,140,140)
DrawEllipse(10,30,150,100)
DrawEllipse(25,15,50,100)
DrawEllipse(33,51,123,143)

DrawText(5,3,"Click within an area to flood it with red")
UseBrush(SOLID,255,0,0)    {Use a red brush for flooding}
UseCoordinates(PIXEL)    {More accurate than metric}

Flood_Wait:
    WaitInput()

Flood_Mouse:
    DrawFlood(Flood_x,Flood_y,0,0,255) {Flood till it meets blue}
    Goto Flood_Wait
```

Related Commands:

[DrawFloodExt](#), [UseBrush](#), [UseBrushPattern](#)

DrawFloodExt

This command is similar to DrawFlood except that it offers an option to continue filling outward in all directions as long as *r,g,b* is encountered.

Syntax: `DrawFloodExt(x,y,r,g,b,BORDER/SURFACE)`

Parameters:

| | |
|----------------|--|
| <i>x</i> | The x-coordinate of a point inside the area you want to fill. |
| <i>y</i> | The y-coordinate of a point inside the area you want to fill. |
| <i>r,g,b</i> | When used with BORDER , specifies the color of the boundary up to which PiXCL is to fill. When used with SURFACE , specifies the color that must be encountered for PiXCL to continue filling. |
| BORDER | Causes PiXCL to fill until the border specified by the <i>r,g,b</i> arguments is reached. Using this setting makes <i>DrawFloodExt</i> identical to <i>DrawFlood</i> . |
| SURFACE | Causes PiXCL to continue filling outward in all directions as long as <i>r,g,b</i> is encountered. |

Example:

The following program draws an ellipse within a rectangle using a green brush and a black pen. If you then click on one of the green areas generated by these shapes, PiXCL uses the *DrawFloodExt* with the SURFACE parameter to change the color of the chosen area to blue.

```
{Set up environment}
  UseCoordinates(PIXEL)  {Coordinate system to pixels}
  r=0
  g=255
  b=0
  UseBrush(SOLID,r,g,b)  {Green brush}
  UsePen(SOLID,1,0,0,0)  {Black pen}

{Draw ellipse within rectangle}
  DrawRectangle(1,1,300,200)
  DrawEllipse(1,1,300,200)

  DrawText(10,215,
           "Click on a green area to change it blue")

{Get mouse point}
  SetMouse(0,0,3000,3000,MouseHit,x,y)

Wait_for_Input:
  WaitInput()

MouseHit:
  UseBrush(SOLID,0,0,255)  {Blue brush}
  DrawFloodExt(x,y,r,g,b,SURFACE)  {Fill until not green}
  Goto Wait_for_Input
```

Related Commands

[DrawFlood](#), [UseBrush](#), [UseBrushPattern](#)

DrawFocusRectangle

Draws a rectangle using a black dotted line in the style that indicates the focus is in the defined rectangle. You specify the upper-left corner of the rectangle using $(x1,y1)$ and the lower-right corner using $(x2,y2)$.

Syntax: `DrawFocusRectangle(x1,y1,x2,y2)`

Parameters:

`x1,y1` The upper-left corner of the rectangle.

`x2,y2` The lower-right corner of the rectangle.

Remarks:

The `DrawFocusRectangle` command toggles, that is, if the same coordinates are issued, the focus rectangle previously drawn at those coordinates is deleted. If a different rectangle is specified, an additional rectangle is redrawn. If your program needs to move the focus rectangle around, you should delete the first rectangle before drawing the next.

Examples:

The following example draws a rectangle whose upper-left corner is located at (10,10) and lower-right corner is at (200,100), waits then turns the focus rectangle off.

```
UseCoordinates (PIXEL)
DrawDocusRectangle (10,10,200,100)
WaitInput (800)
DrawDocusRectangle (10,10,200,100)
```

Related Commands:

[UsePen](#), [UseBrush](#), [SetMouse](#) commands

DrawFpNumber

Syntax: `DrawFpNumber(x,y,Number&,Digits)`

Parameters:

| | |
|--------------------|---|
| <i>x</i> | Specifies the x-coordinate of the starting point of the real. |
| <i>y</i> | Specifies the y-coordinate of the starting point of the real. |
| <i>Number&</i> | The real or the value of the real variable you want to display. |
| <i>Digits</i> | The number of decimal place digits to draw. |

Examples:

This example displays the number -2482.2887 starting 20 pixels to the right and 10 pixels below the upper-left corner of the window.

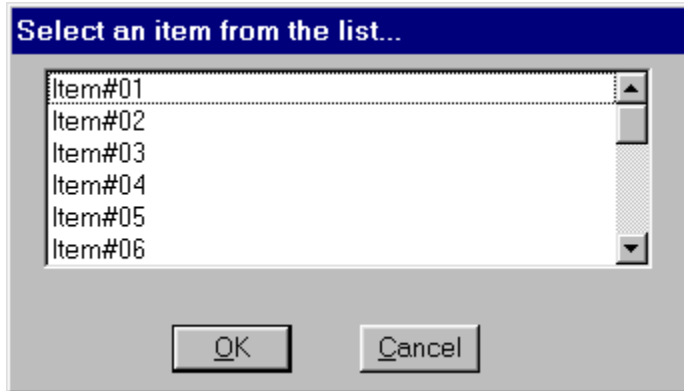
```
Number& = -2482.2887
DrawFpNumber(20,10,Number&,5)
WaitInput()
```

Related Commands:

[DrawNumber](#)

DrawFrameControl

FrameControls are the little bitmaps that are used to create controls in a window frame such as title bar buttons, scrollbar buttons, grips, as well as radio buttons and push buttons. The DrawFrameControl function draws a frame control of the specified type and style, within the specified drawing rectangle.



How can the controls be used ?

Draw the frame control in the desired location and use any of the five SetMouse commands. A [SetMouse](#) handler can redraw the control to indicate that the control has been pushed or activated, and you construct your program to process accordingly.

Syntax: `DrawFrameControl(x1,y1,x2,y2,TYPE_token, STATE_token ,Result)`

Parameters:

`x1,y1,x2,y2`

The client area coordinates of the bounding rectangle for the frame control. This means that the control can be drawn any size you want. The same co-ordinates should be used for an accompanying SetMouse command.

`TYPE_token`

Specifies the type of frame control to draw. This parameter can be one of the following values:

| Value | Meaning |
|-----------------------------|------------------------------|
| <code>BUTTON3STATE</code> | Three-state button |
| <code>BUTTONCHECK</code> | Check box |
| <code>BUTTONPUSH</code> | Push button |
| <code>BUTTONRADIO</code> | Radio button |
| | |
| <code>CAPTIONCLOSE</code> | Close button |
| <code>CAPTIONHELP</code> | Windows 95 only: Help button |
| <code>CAPTIONMAX</code> | Maximize button |
| <code>CAPTIONMIN</code> | Minimize button |
| <code>CAPTIONRESTORE</code> | Restore button |
| | |
| <code>MENUARROW</code> | Submenu arrow |
| <code>MENUBULLET</code> | Bullet mark |
| <code>MENUCHECK</code> | Check mark |
| | |
| <code>SCROLLCOMBOBOX</code> | Combo box scroll bar |
| <code>SCROLLDOWN</code> | Down arrow of scroll bar |

| | |
|-----------------------|--|
| SCROLLLEFT | Left arrow of scroll bar |
| SCROLLRIGHT | Right arrow of scroll bar |
| SCROLLSIZEGRIP | Size grip in bottom-right corner of window |
| SCROLLUP | Up arrow of scroll bar |

STATE_token Use this to change the visible state of the control

| Value | Meaning |
|-----------------|--|
| CHECKED | Control is checked. |
| FLAT | Control has a flat border. |
| INACTIVE | Control is inactive (grayed, the initial state). |
| MONO | Control has a monochrome border. |
| PUSHED | Control is pushed. |

Result Zero if failed, 1 of succeeded.

DrawGrid

Use this command when you want to draw a standard sized rectangular grid anywhere in the PiXCL client area, for example, when preparing to draw a graph.

Syntax: `DrawGrid(x1,y1,x2,y2,Hcellsize,Vcellsize, r,g,b, NONE | STYLE_1 | STYLE_2)`

Parameters:

`x1,y1,x2,y2` The rectangle in which the grid is to be drawn.
`Hcellsize,Vcellsize` The horizontal and vertical grid cell size.
`r,g,b` The color of the pen that is used.

`NONE` No border is drawn around the grid area.
`STYLE_1` A one pixel wide border is drawn around the grid.
`STYLE_2` A double line border is drawn around the grid.

Remarks:

Setting the pen color does not set the current pen value, which happens with the `UsePen` command. If a `STYLE_1` or `STYLE_2` border is specified, the rectangle drawn will use the current brush. If you need just the grid, insert a `UseBrush(NULL,r,g,b)` command before the `DrawGrid` command, as seen below.

Example:

This code fragment gets the current client area coordinates and draws two overlapping grids.

```
DrawingGrids:  
  DrawBackground  
  WinGetClientRect("", cx1, cy1, cx2, cy2)  
  UseBrush(NULL, 0, 0, 0)  
  DrawGrid(cx1, cy1, cx2, cy2, 22, 22, 255, 255, 255, NONE)  
  UseBrush(SOLID, 255, 255, 255)  
  DrawGrid(100, 100, 400, 300, 50, 50, 128, 0, 0, STYLE_2)  
  Goto Wait_for_Input
```

The visual effect of the above code could be duplicated with a For loop, and a series of `DrawLine` commands, but since this is interpreted code, it would be noticeably slower.

Related Commands:

[DrawLine](#) , [UseBackground](#) , [DrawBackground](#) , [UseBrush](#) , [UsePen](#)

DrawIcon

Draw one of eighteen default PiXCL icons, or one of six system icons at the specified position in the client area, at default or user specified size.

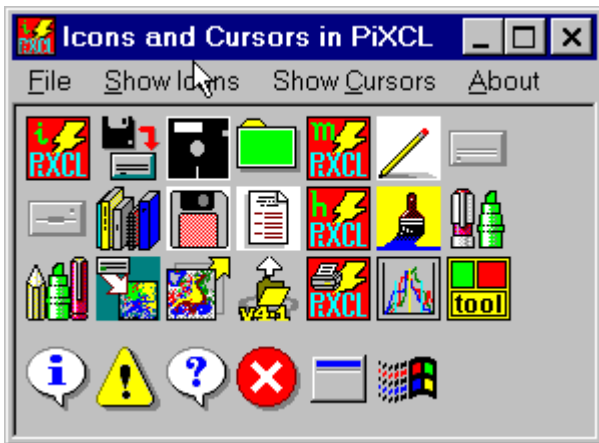
Syntax: `DrawIcon(Xpos, Ypos, Xsize, Ysize, TOKEN)`

Parameters:

- `Xpos, Ypos` The top left corner pixel co-ordinates in the client area where the icon will be drawn.
- `Xsize, Ysize` The size in pixels in which the icon will be drawn. If either of these values are 0, the icon is drawn at the default size of 32x32. The `Xsize, Ysize` values do not have to be the same.
- `TOKEN` This is the upper-case string that defines the icon to be drawn. The twenty-four icons embedded in PiXCL are `ICON01 - ICON19`, `PXLHISTOGRAM` and `PXLTOOLBAR`, `SCANNER`, `DIGICAM` and `SCANCAM`.

The system icon `TOKENS` supported are the same as those available in the `MessageBox(...)` command: `INFORMATION`, `EXCLAMATION`, `QUESTION`, `STOP`, `APP` and `WINLOGO`.

Remarks:



The built-in and system messagebox icons.

The icons above are numbered Left to Right, Top to Bottom.

The built-in icons, `ICON01 - ICON19`, `PXLHISTOGRAM`, `PXLTOOLBAR`, `SCANNER`, `DIGICAM` and `SCANCAM` are also available to be used in the `MessageBox` command.

Example:

Here is some sample code that draws the standard icons, then draws them again.

```
Draw_Icon:
  DrawIcon(10,10,0,0,ICON01)
  DrawIcon(50,10,0,0,ICON02)
  DrawIcon(90,10,0,0,QUESTION)
  DrawIcon(130,10,0,0,EXCLAMATION)
  DrawIcon(170,10,0,0,ASTERISK)
  DrawIcon(210,10,0,0,STOP)
```

It is possible to modify the set of icons in PiXCL and using the `DrawIcon` command, produce an animated icon set. Use an icon management tools to create and insert your own icons into either your runtime applications or into the PiXCL interpreter.

```
DrawIcon(10, 50, 48, 48, ICON01)  
DrawIcon(60, 50, 48, 48, ICON02)  
DrawIcon(110, 50, 48, 48, QUESTION)  
DrawIcon(160, 50, 48, 48, EXCLAMATION)  
DrawIcon(210, 50, 48, 48, ASTERISK)  
DrawIcon(260, 50, 48, 48, STOP)
```

```
Goto Wait_for_Input
```

Related Command:

[DrawBitmap](#), [DrawSizedBitmap](#), [MessageBox](#)

DrawIconFile

This command provides a means to read icons and cursor files from the disk and display them in the client area. The command also supports OPAQUE and TRANSPARENT modes. This is supposed to provide for transparency in icons and cursors. If you need to draw a bitmap with a transparency color, use the [DrawTrBitmap](#) command.

Syntax: [DrawIconFile\(x,y,x_size,y_size,Filename\\$,TOKEN1,TOKEN2\)](#)

Parameters:

| | |
|---|---|
| Xpos , Ypos | The top left corner pixel co-ordinates in the client area where the icon, cursor or bitmap will be drawn. |
| Xsize , Ysize | The size in pixels in which the icon will be drawn. If either of these values are 0, the icon or cursor is drawn at the default size of 32x32. The Xsize , Ysize values do not have to be the same. |
| Filename\$ | The full path and name of the file on disk. |
| TOKEN1 | ICON, CURSOR or BITMAP |
| TOKEN2 | TRANSPARENT or OPAQUE . |

Example:

```
DrawIconFile(60,10,0,0,IconFile$,ICON,TRANSPARENT)
DrawIconFile(110,10,0,0,CursorFile$,CURSOR,TRANSPARENT)
DrawIconFile(10,60,0,0,Bitmap$,BITMAP,TRANSPARENT)
```

Related Commands:

[DrawIcon](#) [DrawTrBitmap](#)

DrawLine

Draws a line using the current pen. The line begins at the point specified by $(x1,y1)$ and extends up to, but does not include, the point specified by $(x2,y2)$.

Syntax: `DrawLine(x1,y1,x2,y2)`

Parameters:

`x1,y1` The coordinates of the first point on the line.

`x2,y2` The coordinates of the ending point on the line.

Example:

This program draws a series of five lines, each one using a different pen width.

```
UsePen (SOLID, 1, 0, 0, 0)
DrawLine (10, 10, 40, 10)
```

```
UsePen (SOLID, 2, 0, 0, 0)
DrawLine (10, 15, 40, 15)
```

```
UsePen (SOLID, 3, 0, 0, 0)
DrawLine (10, 20, 40, 20)
```

```
UsePen (SOLID, 4, 0, 0, 0)
DrawLine (10, 25, 40, 25)
```

```
UsePen (SOLID, 5, 0, 0, 0)
DrawLine (10, 30, 40, 30)
```

```
WaitInput ()
```

See the [UsePen](#) command for another example of DrawLine.

Related Command:

[UsePen](#)

DrawNumber

Displays an integer, *Number*, using the current font. The starting position of the integer is given by the *x* and *y* parameters.

Syntax: `DrawNumber(x,y,Number)`

Parameters:

- x* Specifies the x-coordinate of the starting point of the integer.
- y* Specifies the y-coordinate of the starting point of the integer.
- Number* The integer or the value of the integer variable you want to display.

Examples:

This example displays the number -2482000 starting 20 pixels to the right and 10 pixels below the upper-left corner of the window.

```
Number = -2482000
DrawNumber(20,10,Number)
WaitInput()
```

This next example reads a mouse click and uses the `DrawNumber` command to display the coordinates of where the mouse pointer was when the click took place.

```
{Get screen capacity in millimeters}
  GetScreenCaps(HORZSIZE,Horz)
  GetScreenCaps(VERTSIZE,Vert)

{Set mouse hit-testing area and where to branch on click}
  SetMouse(0,0,Horz,Vert,Draw_Coord,Mouse_x,Mouse_y)
  DrawText(5,5,"Click the mouse anywhere in the window")

Wait_Mouse:
  WaitInput()

{Draw the x coordinate}
Draw_Coord:
  DrawNumber(Mouse_x,Mouse_y,Mouse_x)

{Increase the x coordinate by 4 millimeters for each digit in Mouse_x}
  If Mouse_x>=100 Then x=Mouse_x+12 | Goto Draw_Comma {3 digits}
  If Mouse_x>=10 Then x=Mouse_x+8 | Goto Draw_Comma {2 digits}
  {Else} x=Mouse_x+4 {1 digit only}

{Draw the comma}
Draw_Comma:
  DrawText(x,Mouse_y,",")
{Increase x by 2 and draw the y coordinate}
  x=x+2
  DrawNumber(x,Mouse_y,Mouse_y)
  Goto Wait_Mouse
```

Related Command:

DrawShadowNumber UseFont

DrawNumber64

PIXCL 5.1 command. Displays a 64-bit integer, *Number64*, using the current font. The starting position of the integer is given by the *x* and *y* parameters.

Syntax: `DrawNumber64(x,y,Number64#)`

Parameters:

x Specifies the x-coordinate of the starting point of the integer.

y Specifies the y-coordinate of the starting point of the integer.

Number64# The 64-bit integer **variable** you want to display. This may have been obtained from the [FileGetSize64](#) command.

Related Command:

[FileGetSize64](#) [Str64](#) [Val64](#)

DrawOutlineNumber

PIXCL 5 command. This command extends the [DrawNumber](#) command to draw a number string outlined using the currently defined pen, and filled with the current brush or pattern.

Syntax: [DrawOutlineNumber\(x,y,Number\)](#)

Parameters:

- [x](#) The x-coordinate of the starting point of the number string.
- [y](#) The y-coordinate of the starting point of the number string.
- [Number](#) The number string to be drawn.

Related Command:

[DrawNumber](#) [DrawOutlineText](#) [UsePen](#) [UseBrush](#) [UseBrushPattern](#)

DrawOutlineText

PIXCL 5 command. This command extends the [DrawText](#) command to draw a text string outlined using the currently defined pen, and filled with the current brush or pattern.

Syntax: `DrawOutlineText(x,y,Text$)`

Parameters:

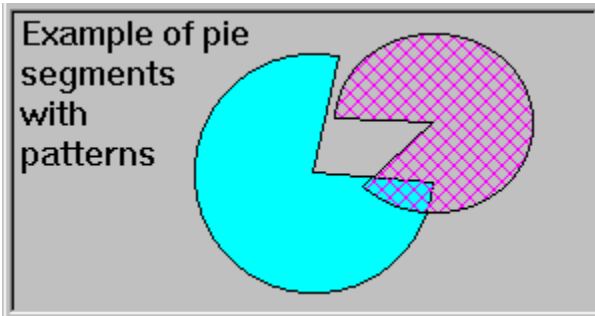
- `x` The x-coordinate of the starting point of the string.
- `y` The y-coordinate of the starting point of the string.
- `Text$` The character string to be drawn.

Related Command:

[DrawText](#) [DrawOutlineNumber](#) [UseBrush](#) [UseBrushPattern](#)

DrawPie

Draws a pie wedge using the current pen and filled with the current brush color and pattern. A pie wedge consists of an arc whose center and end points are connected by lines. To draw the arc, you use the points $(x1,y1)$ and $(x2,y2)$ to define a rectangle that bounds the ellipse containing the arc, as shown in the figure below. You then use the parameters $(x3,y3)$ and $(x4,y4)$ to specify the starting and ending points of the arc. Note that when the GDI sweeps the arc, it begins at $(x3,y3)$ and moves in a counterclockwise direction toward $(x4,y4)$.



DrawPie coordinates

Syntax: `DrawPie(x1,y1,x2,y2,x3,y3,x4,y4)`

Parameters:

- $x1,y1$ The upper-left corner of the rectangle bounding the ellipse containing the arc.
- $x2,y2$ The lower-right corner of the rectangle bounding the ellipse containing the arc.
- $x3,y3$ The starting point of the arc. This point does not have to lie on the arc.
- $x4,y4$ The ending point of the arc. This point does not have to lie on the arc.

Remark:

The points $(x3,y3)$ and $(x4,y4)$ do not have to lie precisely on the ellipse. If they do not, however, the GDI uses points on the ellipse that are the shortest distance from $(x3,y3)$ and $(x4,y4)$.

Examples:

This example draws a pie wedge using the default black pen to draw the border and an aqua brush to fill the interior. The pie wedge is bounded by the rectangle specified by the points (110,30) and (160,80). The pie's arc starts at the point (160,30) and ends at the point (160,80).

```
UseBrush(SOLID,0,255,255)
UseCoordinates(PIXEL)
DrawPie(110,30,160,80,160,30,160,80)
WaitInput()
```

This next script draws a pie wedge based on your mouse clicks. The first two clicks define the rectangle bounding the arc, and the second two define the lines connecting the points of the arc with its center.

```
{Set up the environment}
SetWindow(MAXIMIZE)
UseCoordinates(PIXEL)
UseFont("Terminal",10,10,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
GetScreenCaps(HORZRES,MaxX)
GetScreenCaps(VERTRES,MaxY)
```

Piel:

```

UsePen(DOT,1,0,0,0)      {Dotted pen for temporary ellipse}
UseBrush(NULL,0,0,0)    {Hollow brush for temporary ellipse}
SetMouse(0,0,MaxX,MaxY,Pie2,x1,y1)
Text1$ =
    "Click on upper left corner of rectangle bounding pie "
TextPos = MaxY - 100
DrawText(10,TextPos,Text1$)
Goto Get_Input

Pie2:
SetMouse(0,0,MaxX,MaxY,Pie3,x2,y2)
Text2$ =
    "Click on lower right corner of rectangle bounding pie"
DrawText(10,TextPos,Text2$)
Goto Get_Input

Pie3:
DrawEllipse(x1,y1,x2,y2)
SetMouse(0,0,MaxX,MaxY,Pie4,x3,y3)
Text3$ = "Click on one end of arc defining pie"
Len(Text2$,Length2)
Len(Text3$,Length3)
Difference = Length2 - Length3
Pad(Text3$,Difference)      {Pad text with spaces}
DrawText(10,TextPos,Text3$)
Goto Get_Input

Pie4:
SetMouse(0,0,MaxX,MaxY,Pie_End,x4,y4)
Text4$ = "Click on other end of arc defining pie"
Len(Text4$,Length4)
Difference = Length2 - Length4
Pad(Text4$,Difference) {Pad text with spaces}
DrawText(10,TextPos,Text4$)
Goto Get_Input

Pie_End:
DrawBackground
UsePen(SOLID,1,0,0,0)
UseBrush(SOLID,255,0,0)
DrawPie(x1,y1,x2,y2,x3,y3,x4,y4)
Goto Pie1

Get_Input:
WaitInput()

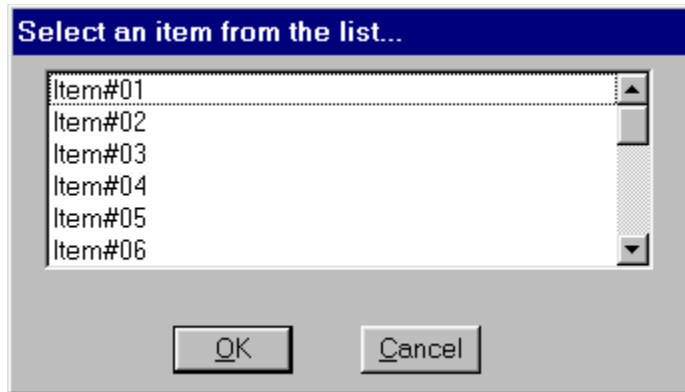
```

Related Commands:

[UsePen](#), [UseBrush](#)

DrawPoint

PiXCL 4.12 and later provides preset point icons that are useful when plotting graphs. The current pen is used to draw the point style. Available point styles are shown in the image below.



Syntax: `DrawPoint(x,y,STYLE_token)`

Parameters:

| | |
|----------------------|---|
| <code>x,y</code> | The desired point coordinates. This may be the return values from a SetMouse command. |
| <code>CROSS</code> | an X, as shown above. |
| <code>PLUS</code> | a + , as shown above. |
| <code>CIRCLE</code> | a circle with a center dot, as shown above. The current brush affects the fill color. |
| <code>BOX</code> | a box with a center dot, as shown above. The current brush affects the fill color. |
| <code>DIAMOND</code> | a diamond with a center dot, as shown above. |
| <code>DOT</code> | a 3x3 dot, as shown above. |

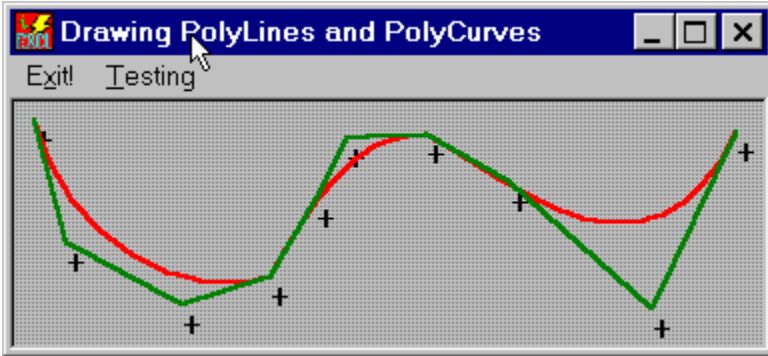
Related Commands:

[UseBrush](#) [UsePen](#)

DrawPolyCurve, DrawPolyLine

[DrawPolyCurve](#) and [DrawPolyLine](#) are variable argument length functions. [DrawPolyCurve](#) draws cubic Bézier curves by using the endpoints and control points specified in the argument list. The first curve is drawn from the first point to the fourth point by using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point. A cubic Bezier curve does not link the points, but calculates and draws the best fit curve.

[DrawPolyLine](#) draws connecting lines between the endpoints and control points specified in the argument list. The figure below shows the difference between [DrawPolyLine](#) (in green) and [DrawPolyCurve](#) (in red) using the same points. Note that the points (“+”), shown for clarity only, are offset because the actual draw points are the top left corner of the character cell.



Both these functions draw lines by using the current pen.

Syntax: [DrawPolyCurve](#)($x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, \dots$)

$x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4$ The first four points that define the Bezier curve. You can specify up to 32 coordinate pairs.

Syntax: [DrawPolyLine](#)($x_1, y_1, x_2, y_2, \dots$)

$x_1, y_1, x_2, y_2, \dots$ The first points that define the polyline. You can specify up to 32 coordinate pairs.

Remarks:

For [DrawPolyCurve](#) the number of points specified must be a multiple of three, plus one (the starting point) or a syntax error will occur. Points can be read in from a file or acquired from the client area using one of the [SetMouse](#) commands.

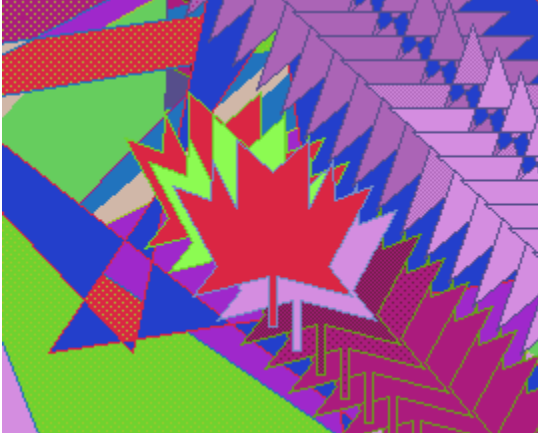
For [DrawPolyLine](#) the minimum number of points is two.

Related Commands:

[DrawLine](#) , [DrawPolygon](#) , [DrawTriangle](#) , [SetMouse](#)

DrawPolygon

Convex and concave polygons with up to 32 vertices can be drawn with the current pen and current brush. [DrawPolygon](#) is a variable argument list command, so you need to include the list of required vertices only. The polygon is drawn in the order that the vertices are specified. The sample image below shows replicated polygons. See sample program **boxes.pxl** for more information. The maple leaf polygon (see image below) in the **boxes.pxl** example has 19 vertices.



Syntax: `DrawPolygon(x1,y1,x2,y2,x3,y3, . . . ,x32,y32)`

Parameters:

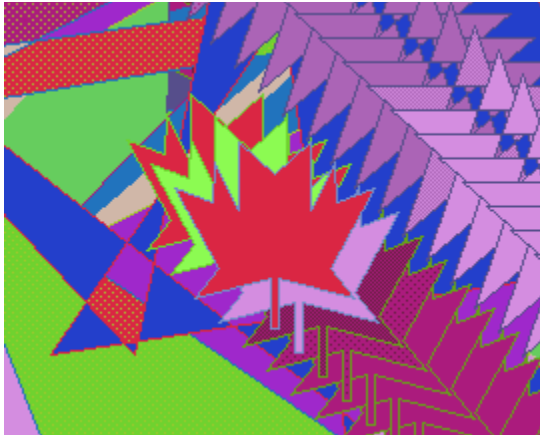
- | | |
|----------------------|---|
| <code>x1,y1</code> | The first vertex of the polygon. |
| <code>x2,y2</code> | The second vertex of the polygon. |
| <code>x32,y32</code> | The thirtysecond vertex of the polygon. |

Related Commands:

[DrawTriangle](#), [DrawRectangle](#), [UsePen](#), [UseBrush](#)

DrawPolygonExt

PIXCL 5 command. Convex and concave polygons with up to 65,536 vertices can be drawn with the current pen and current brush. The polygon is drawn in the order that the vertices are specified. The sample image below shows replicated polygons. See sample program **boxes.pxl** for more information. The maple leaf polygon (see image below) in the **boxes.pxl** example has 19 vertices.



Syntax: `DrawPolygonExt(NumberOfVertices, IntegerArrayVariable[0], Result)`

Parameters:

NumberOfVertices The number of vertices in the polygon.

IntegerArrayVariable[0] An array of points starting at element 0, in order **x0,y0,x1,y1,x2,y2**,... The array must have at least *NumberOfVertices* times 2 elements or the command will have no effect.

Result 1 if the operation was successful, otherwise 0.

Related Commands:

[DrawTriangle](#), [DrawRectangle](#), [UsePen](#), [UseBrush](#)

DrawRectangle

Draws a rectangle using the current pen and fills its interior using the current brush. You specify the upper-left corner of the rectangle using $(x1,y1)$ and the lower-right corner using $(x2,y2)$.

Syntax: `DrawRectangle(x1,y1,x2,y2)`

Parameters:

`x1,y1` The upper-left corner of the rectangle.

`x2,y2` The lower-right corner of the rectangle.

Examples:

The following example draws a rectangle whose upper-left corner is located at (10,10) and lower-right corner is at (200,100). It uses a black pen that is three pixels wide and fills the interior using an orange brush.

```
UsePen (SOLID,3,0,0,0)
UseBrush (SOLID,255,128,0)
UseCoordinates (PIXEL)
DrawRectangle (10,10,200,100)
WaitInput ()
```

This next example draws a rectangle based on two mouse clicks. The first mouse click defines the upper-left corner of the rectangle and the second one defines the lower-right. The rectangle is filled using a light cream-colored brush.

```
{Set up the environment}
  SetWindow (MAXIMIZE)
  UseCoordinates (PIXEL)
  UsePen (SOLID,3,0,0,0)
  UseBrush (SOLID,255,255,230)

UseFont ("Terminal",10,10,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
GetScreenCaps (HORZRES,MaxX)
GetScreenCaps (VERTRES,MaxY)

Rect1:
  SetMouse (0,0,MaxX,MaxY,Rect2,x1,y1)
  TextPos = MaxY - 100
  DrawText (10,TextPos,
    "Click on upper left corner of the rectangle ")
  Goto Get_Input

Rect2:
  SetMouse (0,0,MaxX,MaxY,Rect3,x2,y2)
  DrawText (10,TextPos,
    "Click on lower right corner of the rectangle")
  Goto Get_Input

Rect3:
  DrawRectangle (x1,y1,x2,y2)
  Goto Rect1

Get_Input:
  WaitInput ()
```

Related Commands:

[UsePen](#), [UseBrush](#)

DrawRoundRectangle

Draws a rectangle with rounded corners. The border of the rectangle is drawn using the current pen and the interior is filled using the current brush. You specify the upper-left corner of the rectangle using $(x1,y1)$ and the lower-right corner using $(x2,y2)$ as shown in the figure below. You control the width and height of the ellipse used to draw the rounded corners using $x3$ and $y3$.



DrawRoundRectangle coordinates

Syntax: `DrawRoundRectangle(x1,y1,x2,y2,x3,y3)`

Parameters:

- | | |
|---------|---|
| $x1,y1$ | The upper-left corner of the rectangle. |
| $x2,y2$ | The lower-right corner of the rectangle. |
| $x3$ | The width of the ellipse used to draw the rounded corners. |
| $y3$ | The height of the ellipse used to draw the rounded corners. |

Example:

The following example draws a rounded rectangle whose upper-left corner is located at (20,30) and lower-right corner is at (300,200). The height and width of the ellipse used to draw the rectangle's rounded corners are both 30 pixels. A black pen two pixels wide is used to draw the rectangle's outer shape, and the interior is filled using a green brush.

```
UsePen (SOLID, 2, 0, 0, 0)
UseBrush (SOLID, 0, 255, 0)
UseCoordinates (PIXEL)
DrawRoundRectangle (20, 30, 300, 200, 30, 30)
WaitInput ()
```

Related Commands:

[UsePen](#), [UseBrush](#)

DrawShadeRectangle

Draws a rectangle with a color gradient. Gradients are useful in drawing backgrounds.

Syntax: `DrawShadeRectangle(x1,y1,x2,y2,r1,g1,b1,r2,g2,b2, TOPBOTTOM | BOTTOMTOP)`

Parameters:

| | |
|-----------------------|--|
| <code>x1,y1</code> | The upper-left corner of the rectangle. |
| <code>x2,y2</code> | The lower-right corner of the rectangle. |
| <code>r1,g1,b1</code> | The RGB color that the color gradient starts the draw process. |
| <code>r2,g2,b2</code> | The RGB color that the color gradient ends the draw process. |

`TOPBOTTOM | BOTTOMTOP`

The direction in which the color gradient is generated.

Remarks:

This command takes the input co-ordinates and calculates the RGB color increments that will apply for each line in the rectangle, and draws using a pen. The current pen style is not affected by the [DrawShadeRectangle](#) command.

Example:

The current PiXCL application client area is located and a black to blue gradient background is drawn, then a central region is inverted.

```
WinGetClientRect("", cx1, cy1, cx2, cy2)
DrawShadeRectangle(cx1, cy1, cx2, cy2, 0, 0, 0, 0, 0, 255, TOPBOTTOM)
cx1 = 100
cy1 = 100
cx2 -= 100
cy2 -= 100
InvertRectangle(cx1, cy1, cx2, cy2)
```

Related Commands:

[GradientFillRect](#) [InvertRectangle](#), [WinGetClientRect](#)

DrawShadowFpNumber

It is often handy to draw numbers with a so-called drop shadow in another contrasting color. While this can be done by changing the font color with [UseFont](#) and using multiple [DrawFpNumber](#) commands, the [DrawShadowFpNumber](#) command merges this operation into one command.

Syntax: [DrawShadowFpNumber\(x,y,Number&,Digits,R,G,B,XY_Offset\)](#)

Parameters:

| | |
|--------------------|---|
| <i>x</i> | The x-coordinate of the starting point of the number. |
| <i>y</i> | The y-coordinate of the starting point of the number. |
| <i>Number&</i> | The real number to be drawn. |
| <i>Digits</i> | The number of decimal place digits to draw. |
| <i>R,G,B</i> | The font color to be used for the shadow number. |
| <i>XY_Offset</i> | The +ive or -ive offset used for the shadow number |

Remark:

By default, PiXCL uses the System font in black when drawing text. You can change the font with the [UseFont](#) command.

Example:

These two code fragments are functionally equivalent. First, using [DrawFpNumber](#) ...

```
UseFont ("Arial", 11, 23, NOBOLD, NOITALIC, NOUNDERLINE, 0, 0, 0)
DrawFpNumber (11, 31, AppNumber&, 5)
UseFont ("Arial", 11, 23, NOBOLD, NOITALIC, NOUNDERLINE, 255, 255, 0)
DrawFpNumber (10, 30, AppNumber&, 5)
```

and now, using the [DrawShadowFpNumber](#) command ...

```
UseFont ("Arial", 11, 23, NOBOLD, NOITALIC, NOUNDERLINE, 255, 255, 0)
DrawShadowFpNumber (10, 30, AppNumber&, 5, 0, 0, 0, 1)
```

Related Commands:

[UseFont](#), [UseBackground](#), [DrawFpNumber](#) , [DrawShadowTextExt](#), [DrawTextExt](#) [SetFontEscapement](#)

DrawShadowNumber

It is often handy to draw numbers with a so-called drop shadow in another contrasting color. While this can be done by changing the font color with [UseFont](#) and using multiple [DrawNumber](#) commands, the [DrawShadowNumber](#) command merges this operation into one command.

Syntax: `DrawShadowNumber(x,y,Number,R,G,B,XY_Offset)`

Parameters:

| | |
|------------------------|---|
| <code>x</code> | The x-coordinate of the starting point of the number. |
| <code>y</code> | The y-coordinate of the starting point of the number. |
| <code>Number</code> | The integer number to be drawn. |
| <code>R,G,B</code> | The font color to be used for the shadow number. |
| <code>XY_Offset</code> | The +ive or -ive offset used for the shadow number |

Remark:

By default, PiXCL uses the System font in black when drawing text. You can change the font with the [UseFont](#) command.

Example:

These two code fragments are functionally equivalent. First, using [DrawNumber](#) ...

```
UseFont ("Arial", 11, 23, NOBOLD, NOITALIC, NOUNDERLINE, 0, 0, 0)
DrawNumber (11, 31, AppNumber)
UseFont ("Arial", 11, 23, NOBOLD, NOITALIC, NOUNDERLINE, 255, 255, 0)
DrawNumber (10, 30, AppNumber)
```

and now, using the [DrawShadowNumber](#) command ...

```
UseFont ("Arial", 11, 23, NOBOLD, NOITALIC, NOUNDERLINE, 255, 255, 0)
DrawShadowNumber (10, 30, AppNumber, 0, 0, 0, 1)
```

Related Commands:

[UseFont](#), [UseBackground](#), [DrawNumber](#) , [DrawShadowTextExt](#), [DrawTextExt](#) [SetFontEscapement](#)

DrawShadowText

It is often handy to draw text with a so-called drop shadow in another contrasting color. While this can be done by changing the font color with [UseFont](#) and using multiple [DrawText](#) commands, the [DrawShadowText](#) command merges this operation into one command.

Syntax: `DrawShadowText(x,y,Text$,R,G,B,XY_Offset)`

Parameters:

| | |
|------------------------|---|
| <code>x</code> | The x-coordinate of the starting point of the string. |
| <code>y</code> | The y-coordinate of the starting point of the string. |
| <code>Text\$</code> | The character string to be drawn. |
| <code>R,G,B</code> | The font color to be used for the shadow text. |
| <code>XY_Offset</code> | The +ive or -ive offset used for the shadow text. |

Remark:

By default, PiXCL uses the System font in black when drawing text. You can change the font with the [UseFont](#) command.

Example:

These two code fragments are functionally equivalent. First, using [DrawText](#) ...

```
UseFont("Arial",11,23,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(11,31,"PiXCL 4.1 Graphics and Imaging")
UseFont("Arial",11,23,NOBOLD,NOITALIC,NOUNDERLINE,255,255,0)
DrawText(10,30,"PiXCL 4.1 Graphics and Imaging")
```

and now, using the [DrawShadowText](#) command ...

```
UseFont("Arial",11,23,NOBOLD,NOITALIC,NOUNDERLINE,255,255,0)
DrawShadowText(10,30,"PiXCL 4.1 Graphics and Imaging",0,0,0,1)
```

Related Commands:

[UseFont](#), [UseBackground](#), [DrawShadowTextExt](#), [DrawTextExt](#) [SetFontEscapement](#)

DrawShadowTextExt

It is often handy to draw text in a rectangular region with a so-called drop shadow in another contrasting color. While this can be done by changing the font color with [UseFont](#) and using multiple [DrawTextExt](#) commands, the [DrawShadowTextExt](#) command merges this operation into one command.

Syntax: `DrawShadowTextExt(x1,y1,x2,y2,Text$, LEFT | CENTER | RIGHT,R,G,B,XY_Offset)`

Parameters:

| | |
|---------------------|--|
| <code>x1,y1</code> | The top left corner of the starting rectangle of the string. |
| <code>x2,y2</code> | The bottom right corner of the starting rectangle of the string. |
| <code>Text\$</code> | The character string to be drawn. |

LEFT | CENTER | RIGHT

These tokens dictate how the output text is drawn: Left justified, Centered in the rectangle, or right justified.

`R,G,B` The font color to be used for the shadow text.

`XY_Offset` The +ive or -ive offset used for the shadow text.

Remark:

By default, PiXCL uses the System font in black when drawing text. You can change the font with the [UseFont](#) command.

Example:

These two code fragments are functionally equivalent. First, using [DrawTextExt](#) ...

```
UseFont("Arial",11,23,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawTextExt(11,31,121,61,"PiXCL 4.1 Graphics and Image Processing",LEFT)
UseFont("Arial",11,23,NOBOLD,NOITALIC,NOUNDERLINE,255,255,0)
DrawText(10,30,120,60,"PiXCL 4.1 Graphics and Image Processing",LEFT)
```

and now, using the [DrawShadowTextExt](#) command ...

```
UseFont("Arial",11,23,NOBOLD,NOITALIC,NOUNDERLINE,255,255,0)
DrawShadowTextExt(10,30,120,60,"PiXCL 4.1 Graphics and Image Processing",LEFT,0,0,0,1)
```

Related Commands:

[UseFont](#), [UseBackground](#), [DrawShadowTextExt](#), [DrawTextExt](#), [SetFontEscapement](#)

DrawShellIcon

In addition to the icons built into PiXCL, there are around 70 icons in the system file **shell32.dll**. There are slight differences in the icons depending on the version of **shell32.dll** installed by Internet Explorer 4 or 5 or with Windows 98 or 2000.

Syntax: [DrawShellIcon](#) (*x,y,Xsize,Ysize,TOKEN*)

Parameters:

| | |
|-----------------------------|--|
| x,y | The client-area coordinates for the top left corner of the icon to be drawn. |
| Xsize,Ysize | The size of the icon to be drawn. |
| TOKEN | The icon to be drawn. Note that the numbers are not contiguous, and vary slightly depending on the version of shell32.dll that is installed. If an unsupported icon is specified, the command does nothing. Supported token values are SHICON01-47,133-148, 151-161, 165-171, 173 . |

Example:

See the sample program [shoicons.pxl](#)

Related Commands:

[DrawIcon](#) [DrawIconFile](#)

DrawSizedBitmap

Reads the contents of a bitmap (BMP, JIF, JPEG, PCD, PCX, PNG, PPM, PSD, RAS, Raw, RLE, TGA, TIF) file and either stretches or compresses it to fit within a specified rectangle. You indicate the upper-left corner of the rectangle using $(x1,y1)$, and the lower-right corner using $(x2,y2)$. This command will create a mirror image of the bitmap if $(x1,y1)$ is the lower-right instead of the upper-left corner.

Syntax: `DrawSizedBitmap(x1,y1,x2,y2,Filename$)`

Parameters:

| | |
|-------------------|--|
| $x1,y1$ | The upper-left corner of the rectangle. |
| $x2,y2$ | The lower-right corner of the rectangle. |
| <i>Filename\$</i> | The name of a bitmap file (see DrawBitmap for details). Include the path if the bitmap file is not in the current directory. |

Remarks:

You can draw into the foreground or background by using the [SetDrawMode](#) command.

[DrawSizedBitmap](#) creates a mirror image of a bitmap if $x2,y2$ is above or to the left of $x1,y1$ (see the examples).

[DrawSizedBitmap](#) supports 256-color bitmaps. Be aware, however, that this type of bitmap can consume a substantial amount of memory. For example, a bitmap containing an entire screen image of a 1024x768-pixel 256-color SuperVGA is over 768K.

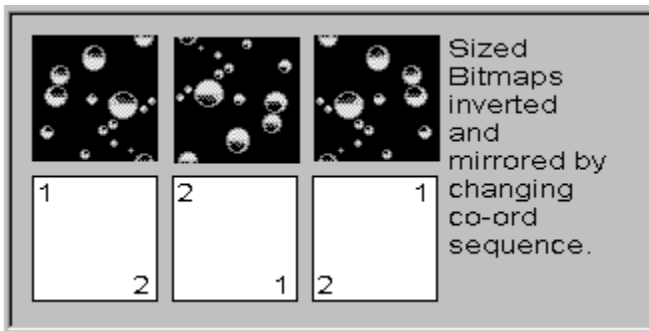
When you use the [DrawSizedBitmap](#) command to place a bitmap in a PiXCL window, PiXCL stores the bitmap image in Windows global memory, as well as in the application memory and display contexts. These contexts are, in effect, the actual PiXCL window as you see it, one stored somewhere in global memory (the memory context), and the other on your video card memory (the display context). When your script ends, PiXCL removes the bitmap(s) from memory and recovers the space it occupies. The image you see drawn in the client area is a copy of the bitmap stored elsewhere in memory.

In most cases, we advise you to remove a bitmap from memory once you have loaded the image file. In most cases, this will not have an undesirable effects. What this approach does is free up the image file global memory, while leaving the memory and display contexts alone.

If you are retrieving the same bitmap from disk and the disk bitmap has changed from the bitmap in program memory, you **MUST** use [FreeBitMap](#) before you reload with [DrawBitmap](#) or [DrawSizedBitMap](#), or else the original bitmap in the memory context will be displayed.

You can also load a bitmap without displaying it by setting all the coordinates to zero. i.e. `DrawSizedBitmap(0,0,0,0,ImageFile$)`, or by using the equivalent [LoadBitmap](#) command.

Examples:



The above image was produced by the following program uses the [DrawSizedBitMap](#) command to read the bubbles bitmap (BUBBLES.BMP) and place it on the screen. (This bitmap is located in the Windows directory.) Next, it uses the [DrawSizedBitMap](#)

command to make additional copies of the bitmap next to the first one and create inverted and mirror images.

```
UseCoordinates (PIXEL) DirGetWindows (WindowsDir$) Bitmap$ = WindowsDir$ + "\
BUBBLES.BMP" UseFont ("Arial", 7, 15, NOBOLD, NOITALIC, NOUNDERLINE, 0, 0, 0)
DrawTextExt (218, 10, 300, 150,
    "Sized Bitmaps inverted and mirrored by changing co-ord sequence.", LEFT)
DrawSizedBitMap (10, 10, 73, 73, Bitmap$)
DrawSizedBitMap (143, 73, 80, 10, Bitmap$)
DrawSizedBitMap (213, 10, 150, 73, Bitmap$)
DrawRectangle (10, 80, 73, 143)
DrawNumber (12, 82, 1) DrawNumber (60, 128, 2)
DrawRectangle (80, 80, 143, 143)
DrawNumber (130, 128, 1) DrawNumber (82, 82, 2)
DrawRectangle (150, 80, 213, 143)
DrawNumber (200, 82, 1) DrawNumber (152, 128, 2)
```

Related Commands:

[DrawBitmap](#), [FreeBitmap](#), [FreeBitmapAll](#), [GetBitMapDim](#), [InvertRectangle](#), [DrawZoomedBitmap](#), [LoadBitmap](#)

DrawSizedBitmapExt

This command offers some additional bitmap draw options according to the TOKEN used. It places the contents of a bitmap (in PiXCL, BMP or .RLE; plus in PiXCLpro, TIF, TGA, RAS, JPEG, PNG and PCX) file at a specified location on the screen.

Syntax: [DrawSizedBitmapExt\(x1,y1,x2,y2,Filename\\$,TOKEN\)](#)

Parameters:

| | |
|----------------------------|---|
| x1, y1 | The coordinates of the upper-left corner of the bitmap. |
| x2,y2 | The coordinates of the bottom-right corner of the bitmap. |
| Filename\$ | The name of the bitmap file. Include the path if the bitmap file is not in the current directory. |

[BLACKONWHITE](#)

[COLORONCOLOR](#)

[HALFTONE](#)

[WHITEONBLACK](#)

Remarks:

These commands will often appear to have no or little effect. See what happens if you use the [SetROPcode](#) command. You can draw into the foreground or background by using the [SetDrawMode](#) command.

Related Commands:

[DrawSizedBitmap](#), [FreeBitmap](#), [FreeBitmapAll](#), [GetBitMapDim](#), [SetColorPalette](#), [DrawZoomedBitmap](#), [LoadBitmap](#)

DrawStatusText

A status text string can be written into a specific region of the PiXCL application client area. The background is the current color, usually (192,192,192). The currently selected font is used. If no font has been selected, the system font is used.

Syntax:

`DrawStatusText(x1,y2,x2,y2, Text$, TOKEN)`

Parameters:

| | |
|--------------------------|---|
| <code>x1,y2,x2,y2</code> | The client area co-ordinates in which the text string is drawn. |
| <code>Text\$</code> | The text string. Carriage returns and Linefeeds are not supported. |
| <code>TOKEN</code> | <code>NOBORDER</code> No raised border is drawn. <code>POPOUT</code> The default. The region appears to be raised. |

Remarks

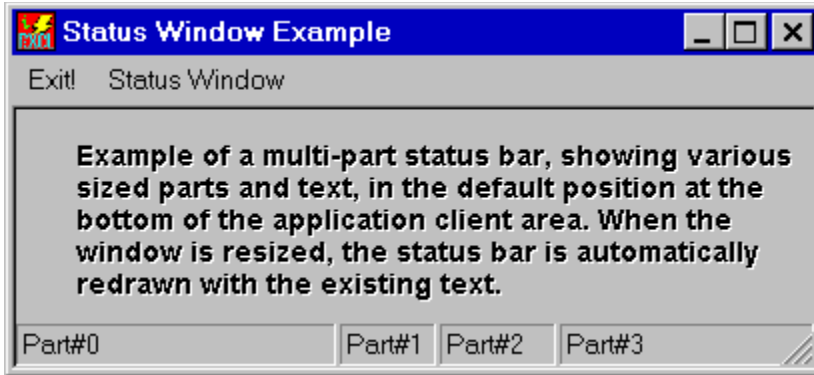
You can also create a status window with the [StatusWindow](#) command, and add text to it with the [DrawStatusWinText](#) command.

Related Commands

[DrawEdgeRectangle](#), [DrawText](#), [DrawTextExt](#), [StatusWindow](#), [DrawStatusWinText](#)

DrawStatusWinText

A status text string can be written into a specific part of a status bar, if it has been enabled. The background is the current system color, usually light gray (192,192,192).



Syntax:

`DrawStatusWinText(Part, StatusText$)`

Parameters:

Part

The zero based index of the required status bar part. If this number is then text is drawn in the first part (which may be the whole status bar). This number must be in the range 0 - 255. Numbers outside this range are clipped to 0 and 255.

StatusText\$

String variable of the required text to be written to the specified part of the status bar.

Related Commands

[DrawTextExt](#), [DrawStatusText](#), [StatusWindow](#).

DrawText

Draws a character string in the window, using the current font. The starting position of the string is given by the **x** and **y** parameters.

Syntax: `DrawText(x,y,Text$)`

Parameters:

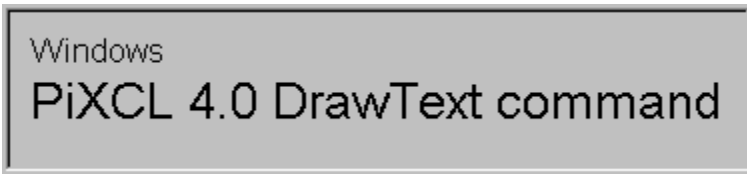
- x** The x-coordinate of the starting point of the string.
- y** The y-coordinate of the starting point of the string.
- Text\$** The character string to be drawn.

Remark:

By default, PiXCL uses the System font in black when drawing text. You can change the font with the UseFont command.

Example:

This example uses Windows Arial font to draw the string "PiXCL 4.0 DrawText command" in a window in two different sizes. The figure shows the results.



The effect of the DrawText command with different size fonts

```
UseCoordinates(PIXEL)
UseFont("Arial",0,20,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,10,"Windows ")
UseFont("Arial",0,30,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,30,"PiXCL 4.0 DrawText command")
WaitInput()
```

This next script shows the effect of the current background setting when you draw text. In this example, the first character string is drawn using the default background, opaque white. The second string is drawn after the background has been set to light gray. Before the third character string is drawn, the font is changed to white, creating a reverse effect. See [UseFont](#) and [UseBackground](#) for more on this.

```
UseCoordinates(PIXEL)
UseFont("System",0,20,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
Name$ = "PiXCL 4.0 DrawText command"
DrawText(10,10,Name$)
UseBackground(OPAQUE,128,128,128) {Light gray background}
DrawText(10,30,Name$)
UseFont("System",0,20,NOBOLD,NOITALIC,NOUNDERLINE,255,255,255)
DrawText(10,50,Name$)
WaitInput()
```

Related Commands:

[UseFont](#), [UseBackground](#), [DrawShadowText](#), [DrawTextExt](#) [SetFontEscapement](#)

DrawTextExt

If you need to draw multiple lines of text, the [DrawText](#) command is rather tedious to use. The [DrawTextExt](#) command draws a multi-line character string in the window client area, in a defined rectangle, using the current font style and color. The coordinates of the rectangle are given by the **x** and **y** parameters.

Syntax: [DrawTextExt\(x1,y1,x2,y2,String\\$,LEFT | CENTER | RIGHT\)](#)

Parameters:

- [x1,y1,x2,y2](#) The rectangle in the client area in which the required text is to be written.
- [String\\$](#) The text that is required to be displayed. This text can include carriage returns, but will also wrap around in the specified rectangle.
- [LEFT | CENTER | RIGHT](#) These tokens dictate how the output text is drawn: Left justified, Centered in the rectangle, or right justified.

Here is an example code fragment that draws a multiline text string on the client area in a specified rectangle.

```
MultiLine:
  TextString$ =
  "This is a multiple line of text that
  should wrap around on the specified
  rectangle region, left justified."
  DrawText(10,30,220,100,TextString$,LEFT)
  Goto Wait_for_Input
```

Related Commands:

[UseFont](#), [DrawText](#), [DrawShadowText](#), [DrawShadowTextExt](#), [SetFontEscapement](#)

DrawTrBitmap, DrawTrSizedBitmap

These two commands extend the bitmap handling capability of PiXCL by providing a means to display a bitmap with a transparency effect, sometimes referred to as a transparent overlay. A specific pixel color, most commonly black, can be specified as the transparency color, such that the underlying or base image is viewable where the overlay image is the transparency color.

Syntax:

`DrawTrBitmap(x1,y1,Image$,R,G,B)`

`DrawTrSizedBitmap(x1,y1,x2,y2,Image$,R,G,B)`

Parameters:

x1,y1, x2,y2 The PiXCL client area co-ordinates for drawing the bitmap.

Image\$ The filename of the image to be drawn. If the image is already in memory, it is drawn from memory, otherwise it is read from the disk. All the supported bitmap file formats are available, and images can be 8 or 24 bits per pixel.

R,G,B The red, green, blue value that defines the transparency color. Values must be in the range 0-255. If the overlay image is 8 bit, the pixel index value that defines *R,G,B* becomes the transparent value. If the bitmap is 24 bit, then the transparency pixels are the *R,G,B* value.

Remarks:

Transparent overlays are often used for simple animation where the transparency color is known, and a sequence of images is to be displayed. The [DrawZoomedBitmap](#) command is also useful in this type of application.

Another use is to show a variety of overlays (perhaps vector information) on a base image, which could for example be a map. The base image and overlay images are loaded and the base image displayed. Each overlay can be displayed in sequence, or on top of each other if desired. The overlay is cleared by redrawing the base image or background.

See also the [OverlayImage](#) command, which works with the current bitmap stored in the PiXCL image list. The contents of this image list can be accessed with the [ListLoadedBitmaps](#) command.

Some supported image formats, for example PNG, provide for a transparency color to be defined within the file itself, but the majority do not. PiXCL 4.1 does not automatically support transparent colors, as the more general solution is to provide support for a transparent display color for any of the supported bitmap formats.

Example:

This code fragment loads two images the same size, and overlays the second image using black (0,0,0) as the transparency value.

```
LoadBitmap(BaseImage$, FULL)
LoadBitmap(OverlayImage$, FULL)
DrawBitmap(10,10,BaseImage$)
WaitInput(2000)
DrawTrBitmap(10,10,OverlayImage$,0,0,0)
```

Remarks:

You can draw into the foreground or background by using the [SetDrawMode](#) command.

Related Commands:

[LoadBitmap](#) , [DrawZoomedBitmap](#) , [SetROPcode](#) , [ChooseColor](#)

DrawTriangle

Triangles can be drawn with the current pen and current brush. [DrawTriangle](#) draws in the order that the vertices are specified.

Syntax: [DrawTriangle\(x1,y1,x2,y2,x3,y3\)](#)

Parameters:

| | |
|-----------------------|------------------------------------|
| x1,y1 | The first vertex of the triangle. |
| x2,y2 | The second vertex of the triangle. |
| x3,y3 | The third vertex of the triangle. |

Related Commands:

[DrawPolygon](#), [DrawRectangle](#), [UsePen](#), [UseBrush](#)

DrawVecPoint, DrawVecLine, DrawVecPolygon and DrawVecLabel

These four commands read in an ASCII format VECtor file into a dynamically allocated memory buffer, and renders the contents into the client area region set in with the [SetVECdrawParams](#) command.

Points are drawn as a box with a center dot. The current brush affects the fill color. Set the current brush to NULL to draw an outline only.

Lines and polygon borders are drawn with the current pen setting defined by the [UsePen](#) command.

Polygons are filled with the current brush setting of the [UseBrush](#) or [UsePatternBrush](#) commands.

Label text is written with the font defined in the current setting of the [UseFont](#) command.

Syntax: [DrawVEC](#)[Point|Line|Polygon|Label] (*VECfilename\$,Result*)

Parameters:

[VECfilename\\$](#) The VECtor file to read into a buffer and render.
[Result](#) 1 if the operation succeeded otherwise 0.

VECtor files in PiXCL:

VEC files are used in PiXCL for the display and transfer of data that is not conveniently representable in raster form. The VEC file formats used in PiXCL are the same as those in the IDRISI™ v2 for Windows GIS, from Clark University in Massachusetts. Idrisi32, released in late 1999, has a slightly updated format, which is not yet supported. It will be supported in PiXCL 5.0, due around July 2000.

VEC files are ASCII files that contain identifiers and coordinates that describe vector features such as points, lines, polygons and text. Any individual VEC file can only contain one type of vector information. A raster image or image set can have any number of VEC files associated with it. The following information is adapted with permission from the IDRISI user manual.

Structure

VEC files in PiXCL are stored as ASCII. While the IDRISI binary VEC format is not supported, IDRISI has a CONVERT module that provides the conversion between ASCII and binary.

In all VEC files, an individual feature (points, lines, polygons and text) is described completely before the next feature is described. Hence a feature is described by

1. a feature identifier;
2. A number of points that defined that feature; and
3. the X and Y coordinates for each point.

A file can contain an arbitrary number of features, but must end with a pair of zeros. If, for example, a simple point file has the first point with an ID of 5 at location 34.5 in X and 76.3 in Y, and the second has an identifier of 3 at 57.3 in X and 12.8 in Y, the file would appear as

```
5      1
34.5   76.3
3      1
57.3   12.8
0      0
```

The next example contains two lines with identifiers 300 and 500, the first with 4 points and the second with 3 points.

```
300    4
21.5   18.1
22.3   21.5
34.1   24.6
```

```
45.9    29.8
500     3
34.5    76.3
64.3    52.1
22.0    12.0
0       0
```

Polygon VEC files are similar to line files, such that the points define the closed polygon, with the first and last entry being the same, which closes the polygon. Hence the example below is for a triangle.

```
110     4
12.2    14.6
56.5    15.3
62.4    85.9
12.2    14.6
0       0
```

IDRISI text files are the same as point files. The actual text labels are stored in an Attribute Values file, another ASCII file, of format.

```
1       label
2       label
3       label
```

For PiXCL, this has been extended with a PiXCL label file, of format

```
ID#1    1 (point)
X       Y (coordinates)
"label Text string 1"
ID#2    1 (point)
X       Y (coordinates)
"label Text string 1"
0       0
```

VEC files can be read and displayed in the PiXCL client area in two ways.

The first is by reading the VEC file contents into a string variable, then using the [DrawPoint](#), [DrawLine](#), [DrawPolygon](#) and [DrawText](#) commands, loop through the coordinates and draw as necessary. While workable, this is also slow, and requires a significant effort in coding.

The alternative method is to use the [DrawVecPoint](#), [DrawVecLine](#), [DrawVecPolygon](#) and [DrawVecLabel](#) commands that read the file into a buffer, interpret and draw the VEC file contents into the client area.

In both cases, the current Pen, Brush and Font color and styles are used to draw the vectors and text. See the [UsePen](#), [UseBrush](#), [UseBrushPattern](#) and [UseFont](#) commands.

VEC files can also be rendered into a loaded image by making use of the [SetDrawBitmap](#) command.

VEC files points can be expressed as either floating point or integer, and to be rendered in the PiXCL client area or a bitmap in the PiXCL image list, the point coordinates have to be converted into client area integer coordinates. For example, if a 500 pixel by 406 line bitmap is displayed full size, starting at client area coordinate (100,80) then we would have to process all the VEC point data with an appropriate gain and offset in both the X and Y axes.

The X and Y gains and offsets, and the client area draw region coordinates are set by the [SetVECdrawParams](#) command, and are stored in the PiXCL executable in memory. The default values are gain=1 and offset=0, draw region is the whole client area. These values remain in force until changed by issuing another [SetVECdrawParams](#) command, in the same manner that the [CustomColors](#) or current font selections are stored.

How and where would VEC files be used?

VEC files are designed to add layered vector information on to an existing raster image, either in the PiXCL image list or directly on to the PiXCL application client area. For example, you can display a color composite, overlay some classification data, and then overlay VEC files that define roads and powerlines, railways, cadastral boundaries, text descriptions, and similar information.

Ascii VEC files can be created by digitizing direct from the client area, using any text editor, or within the IDRISI package.

Related Commands:

[SetVECdrawParams](#) [UsePen](#) [UseBrush](#) [UseBrushPattern](#)

DrawZoomedBitmap

A complete bit map can be displayed with the [DrawBitmap](#) or [DrawSizedBitmap](#) commands. In situations where you want to see just a portion of the bitmap, the [DrawZoomedBitmap](#) command is available. This might be used, for example, by creating a window the dimensions of the bitmap (this assuming that the bitmap is smaller than the screen), and drawing a zoomed region as required.

[DrawZoomedBitmap](#) can also be used to create animated sequences. See the remarks below.

Syntax:

[DrawZoomedBitmap](#)(*x1,y1,x2,y2,Filename\$,pixel_x,pixel_y,ZoomFactor*)

Parameters:

| | |
|---------------------------------|---|
| x1,y1,x2,y2 | The client area coordinates into which the bitmap region is to drawn. |
| FileName\$ | The bitmap to be accessed for pixel data. This does not need to have been previously loaded. |
| pixel_x,pixel_y | The image coordinates on which the zoom occurs. |
| ZoomFactor | A number from 1 to 16. Numbers below or above this range are automatically truncated to 1 or 16 respectively. |

Remarks:

Don't forget that the complete bitmap is stored somewhere in memory, in the linked list that PiXCL keeps and updates with the [DrawBitmap](#) commands, and the display you see written to the client area is a copy of the bitmap, sized if necessary to fit the rectangle specified.

You can also use this command to roam across an image by creating a small window and keeping the zoom factor constant, then move the zoom coordinates along the image. The most useful effect is to choose a line through the horizontal axis.

This command also enables you to view a portion of a bitmap. You will need to use integer math code to set up the [pixel_x,pixel_y](#) coordinates for the selected client area display region at the desired zoom factor.

For example, you can create a large image with an animation sequence tiled in it e.g. like a strip of movie film. Say each tile is 200 lines by 300 pixels, and there is eight frames, then the complete image will be 200 lines x 2400 pixels. The display window would be 200x300, and the zoom factor kept constant at 1. The [pixel_x,pixel_y](#) values are adjusted with a FOR-NEXT or WHILE loop and the effect is animation.

Remarks:

You can draw into the foreground or background by using the [SetDrawMode](#) command.

Related Commands:

[DrawBitmap](#), [FreeBitmap](#), [LoadBitmap](#)

DropFileServer, DropFileServerExt

A PiXCL program can also function as a drop file server for applications that are drop file clients. For example, the FileManager and Explorer are drop file servers, such that you can select a list of filenames, and while holding down the left mouse button, drag them over to a destination window. Windows drop file clients include Write, WordPad, Pbrush, all modern large Microsoft packages that support Object Linking and Embedding, and just about every imaging or drawing program from other software developers. The DropFileServer function is especially useful for Multiple Document Interface (MDI) applications where you need to load up several files at once, and the application supports opening each document sequentially.

For example, a paint and draw program may support multiple image formats, but only lets you open one format file at a time. With the dropfile server function in PiXCL, you can select a variety of images in different formats, and drop them into the application. If it is a dropfile client, it will most likely load up the whole list of images. This can be a substantial saving in time for you.

A program could make use of the [FileRead_INI](#) and [FileWrite_INI](#) commands as well, either with your own INI file for, say, file formats or filters for the [FileGet](#) command, or could even access other applications INI files for information or operating parameters.

The [FileGet](#) dialog also acts as a dropfile server: you can one file (or more files if one of the MULTI select tokens is used) and drag it over to and drop it into a suitable target application.

It's very easy to tell if your application is a drop file client: from Explorer, select a file of the type the application can edit or produce, and try to drag it to and drop it into the application. If the file is automatically loaded, the application is a drop file client.

PiXCL is also a dropfile client, with the [DragAcceptFile](#) and [GetDragList](#) commands.

Syntax: [DropFileServer\(ENABLE | DISABLE,FileList\\$\)](#)
[DropFileServerExt\(ENABLE | DISABLE,IMAGES|FILES|CLIPS|MEDIA,FileList\\$\)](#)

Parameters:

| | |
|----------------------------------|--|
| ENABLE DISABLE | Token that enables or disables the drop file operation. |
| IMAGES | Changes the PiXCL titlebar to " DropFiles – n image(s) to load. " This is the default for DropFileServer . |
| FILES | Changes the PiXCL titlebar to " DropFiles – n files(s) to load. " |
| CLIPS | Changes the PiXCL titlebar to " DropFiles – n clip(s) to load. " |
| MEDIA | Changes the PiXCL titlebar to " DropFiles – n media(s) to load. " |
| FileList\$ | This is a list of at least one filename. It could be generated by a call to FileGet , or read from a file, or synthesized from within the program itself. If the DISABLE token is used, this string can be set to NULL (i.e. "", or say Null\$). A string variable MUST be included, but it is ignored in any case. |

Remarks:

Enabling the drop file server will update the title bar of the PiXCL application with the number of files selected. It is your responsibility to keep track of the title bar string, and update it when needed. You can also draw a special icon on the client area to indicate that the drop file server has some files to process. This is ICON17, built into the PiXCL application.



**Dropfile server icons
16 and 17
which are built into
PiXCL.**

Once you try to drag and drop a filename list, the cursor changes to a monochrome representation of ICON17, and to a cursor when the mouse pointer is over a suitable drop file client.

If the drop file client has successfully loaded the files, the PiXCL title bar will be updated to tell you that there are no files to drop.

If there are any [SetMouse](#) commands active, Drag and Drop enabled overrides it. This can be handy, because once you have completed the drag and drop operation, a [SetMouse](#) command becomes operational, and can be used to clear the drag and drop title.

The code fragment below shows how this might work. An output file has been selected earlier in the program. Once the dropfile server has been enabled, the window title is updated. This is acquired, the current client area coordinates are obtained, and the dropfile icon (ICON17) is drawn at a suitable location. The [SetMouse](#) command is used to disable the [DropFileServer](#), icon and [SetMouse](#) once the dropfile operation is complete.

```
Auto_Load_Image_1:
    DropFileServer (ENABLE, OutFile$)
    WinGetActive (Win$)
    WinGetClientRect (Win$, cx1, cy1, cx2, cy2)
    icx = cx2 - 34
    DrawIcon (icx, 1, 0, 0, ICON17)
    SetMouse (cx1, cy1, cx2, cy2, ClearDFS, X, Y)
    Goto Wait_for_Input
```

```
ClearDFS:
    DropFileServer (DISABLE, "")
    DrawBackGround
    UseCaption (Title$)
    SetMouse ()
    Goto Wait_for_Input
```

Related Commands:

[FileGet](#), [UseCaption](#), [DrawIcon](#), [DragAcceptFile](#), [GetDragList](#)

DuplicateImage

When you want to make a copy of an image in memory, the [DuplicateImage](#) command is used. The new image is added to the PiXCL image list

Syntax: [DuplicateImage\(SourceImageName\\$, NewImageName\\$,Result\)](#)

Parameters:

[SourceImageName\\$](#) The name of the source image in the PiXCL image list.
[NewImageName\\$](#) The duplicated image name. This image is NOT saved to disk.
[Result](#) 1 if the operation is successful, otherwise 0.

Remarks:

[NewImageName\\$](#) should include a full path, as this is how images are searched. Especially, if you expect to save the new image, you will have to fully identify it when using the [SaveBitmap](#) command.

Example:

This code fragment loads a predefined source bitmap, duplicates it with another predefined name, inverts the colors of the new image, then displays them both.

```
ImageDuplicate:  
  DrawBackground  
  LoadBitmap(Image8$,FULL)  
  DuplicateImage(Image8$,Image11$,Res)  
  InvertImage(Res)  
  DrawBitmap(30,38,Image11$)  
  DrawBitmap(230,238,Image8$)  
  Goto Wait_for_Input
```

Related Commands

[EnlargeImage](#) [EnlargeImageBox](#)

EmptyRecycleBin

For system with Internet Explorer 4.01 or later installed, the Recycle Bin(s) can be emptied under program control.

Syntax: [EmptyRecycleBin\(Drive\\$,mode_TOKEN,Result\)](#)

Parameters:

| | |
|----------------------------|---|
| Drive\$ | The drive on which the Recycle Bin resides. If this is an empty string "", all Recycle Bins on all drives are emptied. |
| Mode_TOKEN | Depending on the mode specified, interaction dialogs are presented as the recycle bins are emptied. ALL – Confirm, Progress and sound are used. NOCONFIRM – Disable the confirm dialog. NOPROGRESS – Disable the progress dialog. NOCONFIRMorPROGRESS – disable confirm and progress. NOCONFIRMorSOUND - disables confirm and sound. NONE – just do it with no interaction. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[QueryRecycleBin](#) [DrawShellIcon](#)

End

Terminates a PiXCL program. You can use [End](#) anywhere in a program, either in the main routine or in subroutine. If PiXCL encounters an [End](#) in a subroutine, it will terminate execution not only for the subroutine but for the entire PiXCL program as well. All memory assigned for strings, variables, toolbars, toolwindows, dialogs and bitmaps are freed and returned to Windows.

Syntax: [End](#)

Remark:

The keywords [Quit](#) and [Stop](#) are also accepted, and are synonyms for [End](#). PiXCL also ends a program when it encounters an end-of-file or CTRL-Z character (1AH).

EnlargeImage

When you need to make the size of a bitmap bigger, and place the existing image data at a particular point in the new, larger image, (i.e. add a border) use the [EnlargeImage](#) command.

Syntax: [EnlargeImage](#)(*ImageFile\$,NewLines,NewPixels,PLACEMENT_token,X,Y,Result*)

Parameters:

| | |
|---------------------------------|---|
| ImageFile\$ | The image that has been previously loaded into the image list. |
| NewLines | The new number of lines in the bitmap. |
| NewPixels | The new number of pixels per line. |
| PLACEMENT_token | Position for inserting the original image. |
| CENTER | original image is centered in the new image. |
| TLC | original image is set to TL corner in the new image. |
| TRC | original image is set to TR corner in the new image. |
| BLC | original image is set to BL corner in the new image. |
| BRC | original image is set to BR corner in the new image. |
| USER | TL corner of the original image is set to X,Y in the new image. |
| X,Y | The TL corner placement coordinated for the USER token, otherwise values are ignored. |
| Result | 1 if the operation succeeded otherwise 0. |

Example:

```
EnlargeImageCmd:
  If ImageFile$ <> ""
    GetListBitmapDim(ImageFile$,Lines,Pixels,Bits)
    NewLines = Lines + 20
    NewPixels = Pixels + 20
    EnlargeImage(ImageFile$,NewLines,NewPixels,CENTER,5,5,Res)
    If Res = 0 Then Beep
    DrawBitmap(30,38,ImageFile$)
    GetListBitmapDim(ImageFile$,NewLines,NewPixels,Bits)
    If NewLines = Lines Then DrawStatusWinText(0,"EnlargeImage failed")
  Endif
  Goto Wait_for_Input
```

Related Commands:

[EnlargeImageBox](#)

EnlargeImageBox

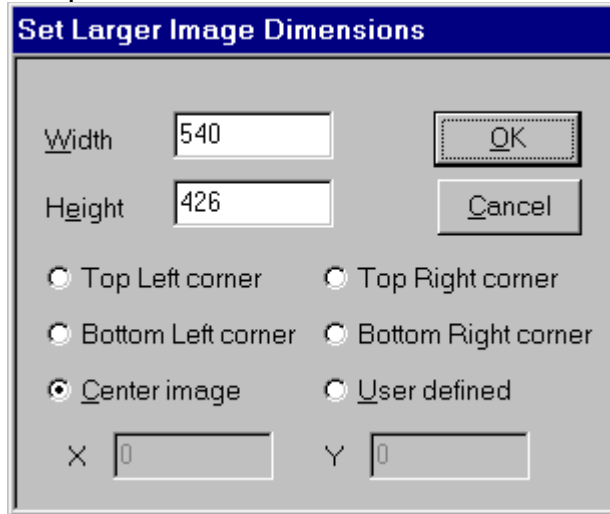
When you want to control the actual setting of an image enlargement operation, PiXCL 4.20 and later provides the EnlargeImageBox command. The dialog is shown below.

Syntax: `EnlargeImageBox(Title$,ImageFile$,Result)`

Parameters:

Title\$ The title that appears in the dialog.
ImageFile\$ The image in the list that is to be enlarged.
Result 1 if the operation succeeded, otherwise 0.

Example:



This code fragment starts the dialog box shown left with a user defined title. Once the OK button is pressed, the image is enlarged. If the Cancel button is pressed, no changes are made. You must issue a [DrawBitmap](#) command to view the enlarged image.

```
EnlargeImageDlg:
  If ImageFile$ <> ""
    EnlargeImageBox("Set Larger Image Dimensions",ImageFile$,Res)
    If Res = 0
      MsgBox(OK,1,ICON01,"EnlargeImageBox failed or was cancelled.",
        "Error Message in user program",Res)
    Else
      DrawBitmap(30,38,ImageFile$)
    Endif
  Endif
  Goto Wait_for_Input
```

Related Commands:

[EnlargeImage](#)

EnumDisplayMonitors

For Windows 98, 2000 and XP systems that have multiple monitors installed, you can list the names and virtual screen coordinates with the [EnumDisplayMonitors](#) command.

Syntax: `EnumDisplayMonitors(m1x1,m1y1,m1x2,m1y2, Monitor_1_Name$,
m2x1,m2y1,m2x2,m2y2, Monitor_2_Name$)`

Parameters:

`m1x1,m1y1,m1x2,m1y2` The virtual screen coordinates of the primary display.

`Monitor_1_Name$` The display name. This is usually `\\.\Display1`

`m2x1,m2y1,m2x2,m2y2` The virtual screen coordinates of the secondary display. Note that if this not the primary display, the coordinates returned may be negative.

`Monitor_2_Name$` The second display name, usually `\\.\Display2`

Remarks:

Many laptops have so-called dual head displays such that the primary display is called `\\.\Display1\Unit0` and the secondary display is `\\.\Display1\Unit1`. Matrox G450/500 cards with the dual head option report the primary display as `\\.\Display1\U0` and the secondary display as `\\.\Display1\U1`. The multi monitor support in Windows defines up to 9 possible displays. If you need to report more than two displays, please contact VYSOR Technical support and we will arrange to update PiXCL.

Related Commands:

[MonitorFromPoint](#) [MonitorFromRect](#) [MonitorFromWindow](#)

EnumChildWindows

When you need to get the names of child windows of any parent window, use this command. It is similar to the [EnumWindows](#) command.

Syntax: `EnumChildWindows(Parent$,Child$,VISIBLE | ALL, Delimiter$)`

Parameters:

| | |
|--------------------|---|
| Parent\$ | The name of the parent window. If you use "", this will find the names of any child windows of the PiXCL application. |
| Child\$ | The list of located child windows. |
| VISIBLE | Token: get only the visible windows. |
| ALL | Token: Get all child windows. |
| Delimiter\$ | The list delimiter character. |

Example:

```
GetChildWindows:
    DrawBackground
    EnumChildWindows("",Child$,VISIBLE,"|")
    UseFont("Arial",0,15,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
    ListBox("Child Windows",Child$,"|",Res$)
    Goto Wait_for_Input
```

Related Commands:

[EnumWindows](#)

EnumPrinters

Basic name information about Local and network / remote printers can be listed. The functionality varies slightly between Windows 95/98 and Windows NT.

Syntax: EnumPrinters(Type_TOKEN,Share_TOKEN,Name\$,Level_TOKEN,
Delimiter\$,PrinterNameList\$,
PrinterDescList\$,Result)

Parameters:

Type_TOKEN

- LOCAL** The function ignores the **Name\$** parameter which should be set to "", and enumerates the locally installed printers. Windows 95/98 only: The function will also enumerate network printers because they are handled by the local print provider.
- NAME** The function enumerates the printer identified by **Name\$**. This can be a server, a domain, or a print provider. If **Name\$** is NULL, the function enumerates available print providers.
- DEFAULT** Windows 95/98 only: The function returns information about the default printer.
- CONNECTIONS** Windows NT only: The function enumerates the list of printers to which the user has made previous connections.
- NETWORK** Windows NT only: The function enumerates network printers in the computer's domain. This value is valid only if **Level_TOKEN** is **ONE**.
- REMOTE** Windows NT only: The function enumerates network printers and print servers in the computer's domain. This value is valid only if **Level_TOKEN** is **ONE**.

Share_TOKEN

SHARED or **NONSHARED**. Non-shared devices include local fax devices and some scanners, such as the Visioneer PaperPort series, which are considered by Windows to be print devices. Non-shared mode will also list available printers.

Name\$

If **Level_TOKEN** is **ONE**, **Type_TOKEN** is **NAME**, and **Name\$** is non-NULL, **Name\$** specifies the name of the object to enumerate. This string can be the name of a server, a domain, or a print provider.

If **Level_TOKEN** is **ONE**, **Type_TOKEN** is **NAME**, and **Name\$** is **NULL**, the function enumerates the available print providers.

If **Level_TOKEN** is **ONE**, **Type_TOKEN** is **REMOTE**, and **Name\$** is **NULL**, the function enumerates the printers in the user's domain.

If **Level_TOKEN** is **TWO** or **FIVE** (Windows 95 only), **Name\$** specifies the name of a server whose printers are to be enumerated. If this string is **NULL**, the function enumerates the printers installed on the local machine.

If **Level_TOKEN** is **FOUR** (Windows NT only), **Name\$** should be **NULL**. The function always queries on the local machine.

When **Name\$** is **NULL**, it enumerates printers that are installed on the local machine. These printers include those that are physically attached to the local machine as well as remote printers to which it has a network connection.

Level_TOKEN

This returns slightly different information to the PiXCL application. The printer names and description or ports are returned.

- ONE** Windows 95/98 and NT
- TWO** Windows 95/98 and NT
- FOUR** Windows NT only

FIVE Windows 95/98 only

Delimiter\$ The character that is used to delimit the returned **List\$** string. This can be any character, but "|" is recommended because printer names can include space, comma and semi-colon characters.

PrinterNameList\$ The returned printer name list.

PrinterDescrList\$ The returned printer description list.

Result 1 if the function succeeded, otherwise 0.

Remarks

The **EnumPrinters** function does not retrieve security information.

Windows NT specific:

Level_TOKEN set to **ONE** provides an easy and extremely fast way to retrieve the names of the printers installed on a local machine, as well as the remote connections that a user has established. When **EnumPrinters** is called with **Level_TOKEN** set to **FOUR**, the function queries the registry for the specified information, then returns immediately. This differs from the behavior of **EnumPrinters** when called with level **ONE**.

In particular, when **EnumPrinters** is called with a **Level_TOKEN** set to **TWO**, it performs an OpenPrinter library call on each remote connection. If a remote connection is down, or the remote server no longer exists, or the remote printer no longer exists, the function must wait for the call to time out and consequently fail the OpenPrinter call. This can take a while. Passing a **Level_TOKEN** set to **TWO** lets an application retrieve a bare minimum of required information.

Windows 95 specific:

To quickly enumerate local and network printers, use **Level_TOKEN** set to **FIVE**. This causes **EnumPrinters** to query the registry rather than make remote calls, and is similar to using **Level_TOKEN** set to **FOUR** on Windows NT as described in the preceding paragraph.

Examples

The following table shows the **EnumPrinters** output with **Level_TOKEN** set to **ONE**.

In **Name\$** you should substitute an appropriate name for Print Provider, Domain, and Machine. For example, for Print Provider, you could use the name of the Windows NT network print provider: "Windows NT Remote Printers", or the name of the Windows 95 local print provider: "Windows 95 Local Print Provider". To get print provider names, call **EnumPrinters** with **Name\$** set to "".

| Type_TOKEN | Name\$ | Result |
|------------|---|--|
| LOCAL | The Name\$ parameter is ignored. Use "". | All local printers. Windows 95 only: Also enumerates network printers because they are installed locally |
| NAME | "Print Provider" | All domain names. |
| NAME | Windows NT only: "Print Provider!Domain" | All printers and print servers in the computer's domain |
| NAME | Windows NT only: "Print Provider!\\Machine" | All printers shared at \\Machine |
| NAME | Windows NT: Set Name\$ to "" Windows 95: The name of the local machine or the local print provider. | All local printers. Windows 95 only: Also enumerates network printers because they are installed locally. |
| NAME | Set Name\$ to "" | All print providers in the computer's domain. |

Windows NT only:

| | | |
|-------------|--|--|
| CONNECTIONS | The Name\$ parameter is ignored | All connected remote printers. |
| NETWORK | The Name\$ parameter is ignored | All printers in the computer's domain. |
| REMOTE | Name\$ = "" | All printers and print servers in the computer's domain |
| REMOTE | "Print Provider" | Same as NAME above. |
| REMOTE | "Print Provider!Domain" | All printers and print servers in computer's domain, regardless of Domain specified. |

EnumWindows

Creates a delimited list of parent window names.

Syntax: `EnumWindows(WindowList$,VISIBLE|ALL,Delimiter$)`

Parameters:

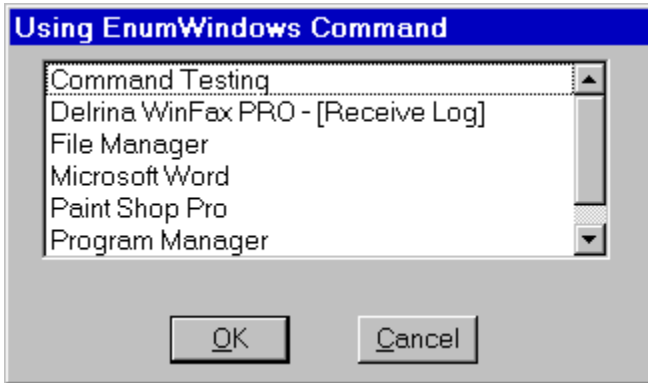
| | |
|---------------------------|--|
| <code>WindowList\$</code> | The string variable to which PiXCL will copy the list of parent windows names. |
| <code>VISIBLE</code> | Directs PiXCL to copy to <code>WindowList\$</code> only the names of visible parent windows (the same names that appear in the Task List). |
| <code>ALL</code> | Directs PiXCL to copy to <code>WindowList\$</code> the names of all parent windows that are open on the Windows 95 and NT desktop, both visible and hidden. |
| <code>Delimiter\$</code> | The delimiter character you want to separate the window names in the list. Many windows titles have spaces in them, so a different character is recommended, such as " " or ";". |

Remarks:

One of the best ways to display the list of window names created by this command is to use the `Listbox` command (see the example).

Example:

This example produces a list of parent window names and then uses the `Listbox` command allowing you to choose a window to activate.



A sample list generated by EnumWindows

```
{Get the name of PiXCL window & hide it}
WinGetActive(PiXCL$)
WinShow(PiXCL$,HIDE,Ignore)
{Show all visible parent windows in a list box}
Delimiter$ = ";"
EnumWindows(WindowList$,VISIBLE,Delimiter$)
Caption$ = "Choose a window to activate"
ListBox(Caption$,WindowList$,Delimiter$,Result$)
{Activate the chosen window}
```

```
If Result$ = "" Then End {User mashed Cancel}
WinSetActive(Result$,Result)
```

Related Command:

[Listbox](#)

EscCommFunction

This commands sends escape functions to the selected COM port. An escape function assists control of the data flow and will change the state of some of the comm port control signals. If a [SetCommPort](#) command has not been previously issued, [EscCommFunction](#) has no effect.

Syntax: [EscCommFunction](#)(COMx, ESC_token)

Parameters:

[COMx](#) Port, where x = 1 – 4 for standard PCs. **PIXCL 5:** If you have a multiple COM port board installed, ports 5-13 are supported.

[ESC_token](#) Specifies the code of the extended function to perform. This parameter can be one of the following values:

Value

Meaning

[CLRDRTR](#)

Clears the DTR (data-terminal-ready) signal.

[CLRRTS](#)

Clears the RTS (request-to-send) signal.

[SETDTR](#)

Sends the DTR (data-terminal-ready) signal.

[SETRTS](#)

Sends the RTS (request-to-send) signal.

[SETXOFF](#)

Causes transmission to act as if an XOFF character has been received.

[SETXON](#)

Causes transmission to act as if an XON character has been received.

[SETBREAK](#)

Suspends character transmission and places the transmission line in a break state until the [CLRREAK](#) extended function code). Note that this extended function does not flush data that has not been transmitted.

[CLRBREAK](#)

Restores character transmission and places the transmission line in a nonbreak state. The [CLRBREAK](#) extended function code is identical to the [ClearCommPort](#) function.

Related Commands:

[ClearCommPort](#) [GetCommPort](#) [ReadCommPort](#) [SetCommPort](#) [WaitCommEvent](#) [WriteCommPort](#)

ExitWindows

This command is the equivalent of selecting the TaskBar **Start:Shutdown** button. It shuts down Windows by closing running applications and prepares the PC to be turned off.

Syntax: `ExitWindows(LOGOFF | POWEROFF | REBOOT | SHUTDOWN)`

Parameters:

| | |
|-----------------------|---|
| <code>LOGOFF</code> | Shuts down all processes running in the security context of the process that called the <code>ExitWindows</code> function. Then it logs the user off. |
| <code>POWEROFF</code> | Shuts down the system and turns off the power. The system must support the power-off feature. Windows NT/2000: The calling process must have suitable privilege. |
| <code>REBOOT</code> | Shuts down the system and then restarts the system. The system must support the power-off feature. Windows NT/2000: The calling process must have suitable privilege. |
| <code>SHUTDOWN</code> | Shuts down the system to a point at which it is safe to turn off the power. All file buffers have been flushed to disk, and all running processes have stopped. Windows NT/2000: The calling process must have suitable privilege. |

Remarks:

Before shutting down, Windows polls all active applications to see if it's OK to close them. If you haven't saved your work, an application can interrupt the shutdown and show a message box like the one in below.

When ExitWindows is executed, active applications may ask you to save your work.

If you have Windows NT network administrator priviledge, you can shut down other PCs on a network by using the Shutdown command.

Example:

The following program places a message box on the screen asking whether you want to shut down Windows. If you select Yes and have saved all your work, Windows shuts down and then displays a restart message.

```
MessageBox(YESNO,1,QUESTION,  
           "Shutdown Windows ?", "Goodbye", Button)  
If Button = 1 Then ExitWindows(REBOOT)
```

Related Commands:

[Logoff](#), [Shutdown](#), [AbortShutdown](#)

Exp

Floating Point math library function. Calculate the exponent function e^{**x}

Syntax: `Exp(X&, Value&)`

Parameters:

`X&` The exponent to use.
`Value&` The result of the function.

Related Commands:

[Log10](#) [LogE](#)

ExportHistogram

Image colour channel histogram details can be exported to an ASCII file with the [ExportHistogram](#) command.

Syntax: [ExportHistogram](#)(*ImageName*,\$,*Result*)

Parameters:

[ImageName](#)\$ An image on disk or loaded into the PiXCL image list.
[Result](#) 1 if the operation succeeded, oitherwise 0.

Remarks:

If [ImageName](#)\$ exists in the PiXCL image list, the histogram is read from memory. If [ImageName](#)\$ is not in memory, and it exists on disk, it is read into memory, the histogram is calculated, then the image is deleted from the list.

The ASCII histogram text file contains the following data, with a CR-LF terminating each line.

Line#001: the image filename;
Line#002: the number of lines in image;
Line#003: the number of pixels per line;
Line#004: the number of BINS (usually 256);
Line#005: the max % of image of any pixel;
Line#006: the max % pixel value;
Line#007-262: the BIN values;
Line#263: mean value;
Line#264: standard deviation.

This file is automatically created, with filename '[<input_filename>.HST](#)' for single channel images. For colour images, three files are created, [<input_filename>R.HST](#)', [<input_filename>G.HST](#)', and [<input_filename>B.HST](#)'.

If the histogram file already exists, it is overwritten.

Related Commands:

[Histogram](#)

ExtractListImageRect

Images loaded in the PiXCL image list can have a rectangle copied into another smaller list bitmap, for example when a set of thumbnail images is to be saved, or to create an animation sequence.

Syntax: [ExtractListImageRect\(SourceImage \\$,DestImage\\$,x1,y1,x2,y2,Result\)](#)

Parameters:

| | |
|-------------------------------|--|
| SourceImage\$ | The (usually larger) source image in the list. |
| DestImage\$ | The smaller image, often created with the CreateBitmap command. If a bitmap of the same name already exists in the list, the contents are overwritten. |
| x1,y1,x2,y2 | The extraction coordinates in SourceImage\$. |
| Result | 1 if the operation was successful, otherwise 0 if the source or destination image does not exist in the list, or -1 if there is a problem with the coordinates. |

Remarks:

The destination image can be saved with the [SaveBitmap](#) command in the desired format. Coordinate problems include x1 or y1 being negative, or x2,y2 outside the boundaries of the source image, or the [DestImage\\$](#) coordinates do not match the size of the rectangle defined in [x1,y1,x2,y2](#). It is your responsibility to ensure that the coordinates specified are correct.

Related Commands:

[CreateBitmap](#) [DuplicateImage](#) [InsertListImageRect](#) [SaveBitmap](#)

FileCopy

Copies one or more files from one directory to another.

Syntax: `FileCopy(SourceName$,DestinationName$,Result)`

Parameters:

- SourceName\$** The name of the file you want to copy. If the file is not located in the current directory, include a path preceding the filename.
- DestinationName\$** The name of the target file. If the target file is to be located outside the current directory, include a path preceding the filename.
- Result** An integer variable that indicates the number of files copied.

Remarks:

PIXCL's FileCopy command observes many of the same shortcuts as the Windows copy command:

- **SourceName\$** and **DestinationName\$** can contain wildcards (? and *).
- You can specify a directory (path with no filename) for **SourceName\$** or **DestinationName\$**. (Unlike previous versions of PIXCL, a "\" doesn't need to be present at the end of the directory name for the operation to be successful.)
- You can use a relative path (for example, "..\temp") or a fully qualified path for **SourceName\$** or **DestinationName\$**.

The **Result** value indicates the number of files copied. If the target filename already exists, the copy command will fail, and **Result** will return 0. Hence, your code should check if the file already exists, and if so either delete it or ask the user for advice.

Example:

This program copies all files with a .TXT extension in the C:\BUDGET directory to the D:\BACKUP directory. If no files are copied, the program beeps and ends. If at least one file is copied, a message box appears signaling success.

```
FileCopy("C:\BUDGET\*.TXT", "D:\BACKUP\", Copy_OK)
If Copy_OK=0 Then Beep | End
MessageBox(OK,1,INFORMATION,"File(s) successfully copied",
          "",Temp)
```

Another example to copy a set of files from one directory to another directory.

```
FileCopy("D:\SOURCE\*.C", "E:\BACKUP\*.\"", Res)
```

If there is, say, ten files in the source directory, **Res** will return a value of 10, and any of the named files existing in the target directory will be overwritten.

```
FileCopy("D:\SOURCE\*.C", "E:\BACKUP", Res)
```

The above command will copy the first *.C file only to the target directory, and **Res** returns 1.

Related Commands:

[DiskChange](#), [FileMove](#), [FileRename](#)

FileDecrypt

Windows 2000 only. The [FileDecrypt](#) command decrypts a file or directory that was previously encrypted. All data streams in a file are decrypted. All new files created in a decrypted directory are decrypted.

Syntax: [FileDecrypt](#)(*FileName*,\$,*Result*)

Parameters:

[FileName](#)\$ The file or directory that is to be decrypted.

[Result](#) 1 if the operation was successful, otherwise 0. If this command is used under Windows 9x or NT, it returns 0, and has no effect.

Related Commands:

[FileEncrypt](#)

FileDelete

Deletes one or more files.

Syntax: `FileDelete(Filename$,Result)`

Parameters:

Filename\$ The name of the file you want to delete. If the file is located outside the current directory or on a different drive, precede the filename with a path.

Result An integer variable that indicates the number of files deleted.

Remarks:

PIXCL's `FileDelete` command observes many of the same shortcuts as Windows `del` command:

- *Filename\$* can contain wildcards (`?` and `*`).
- You can specify a directory (path with no filename) for *Filename\$*. (Unlike previous versions of PIXCL, a `\` doesn't need to be present at the end of the directory name for the operation to be successful.)
- You can use a relative path (for example, `..\temp`) or a fully qualified path for *Filename\$*.

Example:

This program deletes all the files in the `C:\WINDOWS\TEMP` directory. If at least one file is deleted, the program displays a success message.

```
False=0
FileDelete("C:\WINDOWS\TEMP\*",Munched)
If Munched=False Then Beep | End
MessageBox(OK,1,INFORMATION,"File(s) successfully deleted",
           "",Temp)
```

Related Commands:

[DiskChange](#), [FileMove](#), [FileRename](#)

FileEncrypt

Windows 2000 only. The [FileEncrypt](#) command encrypts a file or directory. All data streams in a file are encrypted. All new files created in an encrypted directory are encrypted.

Syntax: [FileEncrypt](#)(*FileName*,\$,*Result*)

Parameters:

[FileName](#)\$ The file or directory that is to be encrypted.

[Result](#) 1 if the operation was successful, otherwise 0. If this command is used under Windows 9x or NT, it returns 0, and has no effect.

Related Commands:

[FileDecrypt](#)

FileExist

Determines whether a file or directory exists on disk.

Syntax: `FileExist(Filename$,Result)`

Parameters:

Filename\$ The name of the file or directory whose existence you want to verify.

Result An integer variable that indicates the number of files/directories found.

Remarks:

Filename\$ can use a relative path or a fully qualified path. It can also contain wildcards (? and *).

This function works with directories as well as files.

Example:

This program tests for the presence of CONFIG.BAK in the root directory of drive C. If the file exists, the variable *There* is set to 1 and the *Test_Overwrite* subroutine is called to verify whether to overwrite the existing copy of CONFIG.BAK. If you select Yes in response to the message box, *Munch* is set to 1 and the subroutine ends; the *FileCopy* command then copies CONFIG.SYS to CONFIG.BAK, overwriting the previous copy of CONFIG.BAK in the process. If you select No, the program ends without performing the copy operation.

```
FileExist("C:\CONFIG.BAK",There)
If There>0 Then Gosub Test_Overwrite
FileCopy("C:\CONFIG.SYS","C:\CONFIG.BAK",Ignore)
End

Test_Overwrite:
MessageBox(YESNO,1,QUESTION,"Overwrite current CONFIG.BAK?",
           "",Munch)
If Munch=Yes Then Return
End
```

Related Commands:

[DiskChange](#), [FileMove](#), [FileRename](#)

FileExtension, FileName, FilePath

From a given full filename, extract either the extension, rootname or path.

Syntax

`FileExtension(FileName$,Extension$,Result)`

`FileName(FileName$,RootName$,Result)`

`FilePath(FileName$,Path$,Result)`

Parameters

| | |
|--------------------|--|
| <i>FileName\$</i> | A complete filename string. This will usually be obtained from a FileGet or similar command, but can also be entered as a string. It must have the form <code>[disk:\path\rootfilename.[extension]</code> with maximum length is 256 characters No check is made if the file actually exists. This can be done if you use one of the "*EXIST" tokens in the FileGet command, or the FileExist command before this command. |
| <i>Extension\$</i> | The returned file extension string. The "." is not included. If the file has no extension, this value returns a NULL string. |
| <i>RootName\$</i> | The returned rootname of the file. If there is no rootname, that is a poorly construct name string like <code>c:\temp\obj</code> , <i>RootName\$</i> returns a NULL string, and <i>Result</i> returns 0. |
| <i>Path\$</i> | The returned path of the file. |
| <i>Result</i> | If the string operation is successful, <i>Result</i> is 1, otherwise it returns 0. If the file does not have an extension (e.g. <code>c:\temp\imagefile</code> .) <i>Result</i> returns 0. If <i>FileName\$</i> is a NULL string, <i>Result</i> returns 0. |

Example

For a *FileName\$* string of `"c:\pixcl\test\imagefile.format"`

| | |
|----------------------------|----------------------------|
| <i>Path\$</i> returns | <code>c:\pixcl\test</code> |
| <i>Name\$</i> returns | <code>imagefile</code> |
| <i>Extension\$</i> returns | <code>format</code> |
| <i>Result</i> returns | 1 |

Here is a code fragment that demonstrates the various types of command and the response.

```
Split_name:
DirGet(SourceDir$)
Filter$ = "All Files(*.*),*.*"
InitFile$ = "*.*"
InitDir$ = SourceDir$
Caption$ = "Select any file"
FileGet(Filter$, InitFile$, InitDir$, Caption$,
    CHANGEDIR_EXIST,FileName$)
If FileName$ = "" Then End

FilePath(FileName$,Path$,Rsl)
```

```
FileName (FileName$,RootName$,Res)  
FileExtension (FileName$,Extension$,Rsl)
```

```
Split$ = Path$ + " "  
Split$ = Split$ + RootName$  
Split$ = Split$ + " "  
Split$ = Split$ + Extension$
```

```
Label$ = "Split " + FileName$  
Res$ = RootName$  
ListBox (Label$,Split$," ",Res$)
```

```
Goto Wait_for_input
```

```
Run_Leave:  
End
```

Related Commands

[FileExist](#), [FileGet](#)

FileGet

Gets a filename using the same [FileOpen](#) common dialog box that appears in most Windows applications.

Syntax: `FileGet(Filter$,InitialFile$,InitialDir$,Caption$,
FileGet_TOKEN,ChosenFile$ [,HelpTitle$, HelpMsg$])`

Parameters:

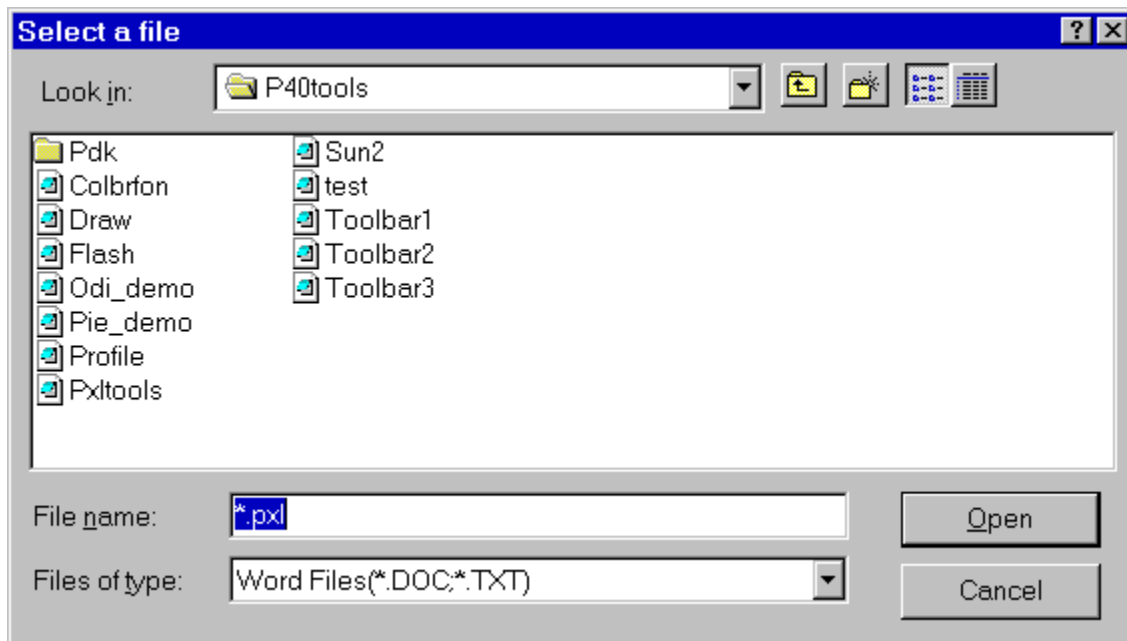
| | |
|---|---|
| <i>Filter\$</i> | A string containing pairs of comma-delimited filter elements. The first element in each pair is text describing the filter (for example, "Write Files"), and the second element specifies the filter pattern (for example, "*.wri"). You can specify multiple patterns for an element by separating them with a semicolon (for example, "*.txt;*.asc;*.pxl"). If you use a null string (" ") for <i>Filter\$</i> , the dialog box will not display any filters. |
| <i>InitialFile\$</i> | A string containing the filename you want to use to initialize the File Name edit control. If you don't want an initial file to appear, specify a null string. |
| <i>InitialDir\$</i> | A string containing the initial directory you want to look at. If you want the current directory to be the initial directory, specify a null string. |
| <i>Caption\$</i> | The text you want to appear in the title bar of the dialog box. If you specify a null string, PiXCL uses "Select a file". |
| <i>FileGet_TOKEN</i> | Determines what happens to the current directory setting on leaving the dialog box and must be one of the following tokens: |
| CHANGEDIR | On exit of the command, change the current disk\directory to the selected directory from the initial directory specified. This will affect the result of a DirGet() command. |
| CHANGEDIRMULTI | Enables file multi-select operations. |
| CHANGEDIR_EXIST | Checks if the file and path exists. An error dialog appears if the file cannot be located. |
| CHANGEDIRMULTI_EXI | Combination of the above. |
| NOCHANGEDIR | If a new disk\directory was selected, do not change the current directory setting. |
| NOCHANGEDIRMULTI | Enables file multi-select operations. |
| NOCHANGEDIR_EXIST | Checks if the file and path exists. An error dialog appears if the file cannot be located. |
| NOCHANGEDIRMULTI_EXI | Combination of the above. |
| Any of the above + <i>_HLP</i> | Displays the dialog with a Help button that displays a Messagebox with custom text and title supplied in <i>HelpTitle\$</i> and <i>HelpMsg\$</i> . |
| PiXCL 5 command. Any of the above + <i>_PREV</i> or <i>_APREV</i> e.g. <i>NOCHANGEDIR_HLP_PREV</i> | Displays the dialog with a Preview and information window. This is useful only if you are previewing supported image formats. <i>_APREV</i> enables the preview mode |

automatically.

- ChosenFile\$** A string variable that will contain the chosen filename, including its path. If the user selects Cancel to leave the dialog, this variable is assigned a null string ("").
- HelpTitle\$** The title of the Help Messagebox.
- HelpMsg\$** The help message that is displayed in the MessageBox.

Example:

This program uses the [FileGet](#) command to prompt you for the name of a word processing file. After you specify a filename and choose OK, the program checks the filename's extension. If it's .PXL, a command line is built to launch Windows Write (for example, WRITE C:\WINDOWS\EXAMPE.WRI). For all other extensions, a command is built to launch Word for Windows.



The FileGet dialog box

```
Filt$ = "PiXCL Files (*.PXL),*.pxl,Write Files (*.WRI),*.wri"  
FileGet(Filt$,"","\windows","",CHANGEDIR,ChosenFile$)  
If ChosenFile$ = "" Then End  
Right(ChosenFile$,3,Ext$) {Get file extension}  
UCase(Ext$) {Convert to uppercase}  
If Ext$ = "WRI" Then CmdLine$ = "write " + ChosenFile$ | Goto Run  
CmdLine$ = "\winword\winword " + ChosenFile$
```

```
Run:  
Run(CmdLine$)
```

Related Commands:

- [DirGetSystem](#) [FileSaveAs](#) [ListBox](#)

FileGetDate

Reads the date stamp in a file's directory entry.

Syntax: `FileGetDate(Filename,$, Year,Month,Day,Result)`

Parameters:

| | |
|--------------------|---|
| <i>Filename</i> \$ | The name of the file whose date you want to read. If the file is located outside the current directory or on a different drive, include a path preceding the filename. |
| <i>Year</i> | A 4 digit integer variable that will contain the year the file was last written. |
| <i>Month</i> | An integer variable that will contain the month the file was last written (1 through 12). |
| <i>Day</i> | An integer variable that will contain the day the file was last written (1 through 31). |
| <i>Result</i> | An integer variable that indicates the outcome of the operation. If the file was found and its date read, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

This command is handy when you are using PiXCL to create an install program and you need to check a file's date before overwriting it with a newer version.

Year, *Month*, and *Day* are returned in the format used in the directory display, but without any preceding zeros.

Example:

This program uses the FileGetDate command to check the date of CMD.EXE file in the Windows system directory. It then displays a message box showing the date in a form similar to 3-5-96.

```
{Get Windows system directory}
  DirGetSystem(SystemDir$)

{Add \CMD.EXE to end of it}
  Cmd$ = SystemDir$ + "\CMD.EXE"

{Get CMD.EXE's date}
  FileGetDate(Cmd$, Year,Month,Day,Result)

{Show date in message box}
  If Result = 1
    TextOut$ = "The file's date is "
    Str(Month,Month$)
    TextOut$ = TextOut$ + Month$
    TextOut$ = TextOut$ + "-"
    Str(Day,Day$)
    TextOut$ = TextOut$ + Day$
    TextOut$ = TextOut$ + "-"
    Str(Year,Year$)
    TextOut$ = TextOut$ + Year$
  Else
    TextOut$ = "File not found"
```

```
Endif
```

Message:

```
MessageBox(OK, 1, INFORMATION, TextOut$, "File date", Ignore)
```

Related Commands:

[FileGetDateExt](#), [FileGetTime](#), [FileGetTimeExt](#)

FileGetDateExt

This function gets a file's date, returning all the day-related statistics offered by the file system.

Syntax:

```
FileGetDateExt(Filename,$CREATION/LASTACCESS/LASTWRITE,  
              Year,Month,DayOfWeek,Day,Result)
```

Parameters:

| | |
|--------------------|---|
| <i>Filename</i> \$ | The name of the file whose date you want to read. If the file is located outside the current directory or on a different drive, include a path preceding the filename. |
| CREATION | Causes PIXCL to return statistics relating to a file's day of creation. |
| LASTACCESS | Causes PIXCL to return statistics relating to the day a file was last accessed. |
| LASTWRITE | Causes PIXCL to return statistics relating to the day a file was last written. |
| <i>Year</i> | An integer variable that will contain the file's year. |
| <i>Month</i> | An integer variable that will contain the file's month (1 through 12). |
| <i>DayOfWeek</i> | An integer variable that will contain the file's day of the week; Sunday = 0, Monday = 1, and similar. |
| <i>Day</i> | An integer variable that will contain the file's day number in the month (1 through 31). |
| <i>Result</i> | An integer variable that indicates the outcome of the operation. If the file was found and its date read, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

In Windows, dates are measured from January 1st, 1601. Statistics are available for the creation date, last-access date, and last-write date. The FAT file system does not record the creation or last-access dates, but both the high-performance file system (HPFS) and the New Technology file system (NTFS) do.

If the file system doesn't record a particular form of date, it is returned as January 1, 1601.

Writing to a file changes the last-write date. Writing to or reading from the file (including running an executable file) changes the last-access date.

Use the [GetVolumeType](#) command to determine the file system used by a drive.

Example:

This example reports the last-write, creation, and last-access dates for a chosen file. The figure below shows the results when the FAT file system is being used for the volume. You would see more extensive statistics for an HPFS or NTFS volume.

| | |
|---------------------|-------------|
| CREATION | |
| Year | 1996 |
| Month | 3 |
| Day of Week | 4 |
| Day of Month | 7 |

FileGetDateExt statistics for a FAT-system file

```

(Get a filename using common dialog)
Set Filt$="All Files (*.*) , *.*"
FileGet(Filt$,"","","Choose a file",CHANGEDIR,ChosenFile$)
If ChosenFile$="" Then End

(Show LASTWRITE date info on screen)
FileGetDateExt (ChosenFile$, LASTWRITE, Year, Month, DayOfWeek,
                Day, Result)
UseFont ("Arial", 7, 15, NOBOLD, NOITALIC, NOUNDERLINE, 0, 0, 0)
DrawText (10, 10, "CREATION")
DrawText (10, 20, "Year")
DrawNumber (65, 20, Year)
DrawText (10, 30, "Month")
DrawNumber (65, 30, Month)
DrawText (10, 40, "Day of Week")
DrawNumber (65, 40, DayOfWeek)
DrawText (10, 50, "Day")
DrawNumber (65, 50, Day)

(If file system is FAT, quit drawing now)
Substr (ChosenFile$, 3, 1, RootDir$)
GetVolumeType (RootDir$, VolumeType$, Ignore)
If VolumeType$ = "FAT" Then Goto Wait_for_input

(Show LASTACCESS date info on screen)
FileGetDateExt (ChosenFile$, LASTACCESS, Year, Month, DayOfWeek,
                Day, Result)
DrawText (90, 10, "LASTACCESS")
DrawText (90, 20, "Year")
DrawNumber (135, 20, Year)
DrawText (90, 30, "Month")
DrawNumber (135, 30, Month)
DrawText (90, 40, "Day of Week")
DrawNumber (135, 40, DayOfWeek)
DrawText (90, 50, "Day")
DrawNumber (135, 50, Day)

(Show CREATION date info on screen)
FileGetDateExt (ChosenFile$, CREATION, Year, Month, DayOfWeek,
                Day, Result)
DrawText (180, 10, "LASTWRITE")
DrawText (180, 20, "Year")
DrawNumber (225, 20, Year)
DrawText (180, 30, "Month")
DrawNumber (225, 30, Month)
DrawText (180, 40, "Day of Week")

```



```
DrawNumber(225,40,DayOfWeek)
DrawText(180,50,"Day")
DrawNumber(225,50,Day)
```

```
Wait_for_input:
  WaitInput()
```

Related Commands:

[FileGetDate](#), [FileGetTime](#), [FileGetTimeExt](#), [GetVolumeType](#)

FileGetExpandedName

This command returns the name of a file that was compressed with the Microsoft utility **compress.exe** with the **-r** flag specified.

Syntax: `FileGetExpandedName(CompressedName%, RealName%)`

Parameters:

CompressedName% The compressed file name ending in an underscore.

RealName% The output file. If the input file is not a compressed file, *RealName*% returns an empty string.

Related Commands:

[FileLZExpand](#)

FileGetSize

Reads the file entry and returns the size of the file in bytes.

Syntax: `FileGetSize(FileName$,Size)`

Parameters:

- FileName\$** The name of the file whose size you want to read. If the file is located outside the current directory or on a different drive, include a path preceding the filename.
- Size** An integer variable that indicates the outcome of the operation. If the file was found, this variable is assigned its size in bytes. Otherwise, it is assigned a value of 0.

Remark:

This command is handy when you are using PiXCL to create a backup program and you need to check on the size of the file.

Example:

This program asks for a file, gets the size, then writes the name and size on to the client area. A destination disk is queried for available space, and if the file is smaller than the available space, it is copied to the destination disk. This example also gets the disk free space.

Initialize:

```
Set NewCaption$ = "Getting File Size"
UseCoordinates (PIXEL)
WinGetActive (OldCaption$)
WinTitle (OldCaption$,NewCaption$)
WinLocate (NewCaption$,200,100,580,310, Res)

SetMenu ("Exit!",Run_Leave,
        ENDPOPOP,
        "Get File",Get_file,
        ENDPOPOP)
```

Wait_for_input:

```
WaitInput ()
```

Get_file:

```
Filter$ = "All Files(*.*),*.*"
InitDir$ = "C:\\"
InitFile$ = "*.*"
Label$ = "Select a file"
FileGet (Filter$,InitDir$,InitFile$,Label$,
        CHANGEDIR_EXIST, File$)
FileGetSize (File$,Size)
DrawBackground
DrawText (10,10,File$)
DrawNumber (10,30,Size)
Disk$ = "D:\\"
GetDiskSpace (Disk$,Type$,Total,Free)
NewFile$ = Disk$ + File$
If Size < Free Then FileCopy (File$,NewFile$,Res)
Goto Wait_for_input
```

Run_Leave:

End

Related Commands:

[FileGetSize64](#) [GetDiskSpace](#)

FileGetSize64

PIXCL 5.1 command. Reads the file entry and returns the size of the file in bytes. This command supports files larger than 2GB. Note that 64 bit integer support in PiXCL 5 is limited. If there is a specific 64 bit command you need, please contact VYSOR Integration Inc. technical support with the details.

Syntax: `FileGetSize64(FileName$,Size64#)`

Parameters:

FileName\$

The name of the file whose size you want to read. If the file is located outside the current directory or on a different drive, include a path preceding the filename.

Size64#

A 64-bit integer variable that indicates the outcome of the operation. If the file was found, this variable is assigned its size in bytes. Otherwise, it is assigned a value of 0.

Related Commands:

[DrawNumber64](#) [Str64](#) [Val64](#)

[FileGetSize](#) [GetDiskSpace](#)

FileGetTempName

It is often necessary to create a temporary filename while an application is running. The [FileGetTempName](#) command provides a simple means to generate a unique temporary name, and a zero length file on your hard disk. PiXCL does not keep a record of these temporary files, nor does Windows. It is your responsibility to delete these temporary files when the application terminates, with the [FileDelete](#) command.

Syntax: [FileGetTempName](#)(*Directory\$,Prefix\$,Unique,TempFile\$*)

Parameters:

| | |
|-----------------------------|---|
| Directory\$ | The directory path for the filename. This string must consist of characters in the ANSI character set. Applications typically specify a period (.) i.e. the current directory, or the result of the GetTempPath command for this parameter. If this parameter is NULL, the command fails. |
| Prefix\$ | A prefix string. The first three characters of this string are used as the prefix of the filename. This string must consist of characters in the ANSI character set. |
| Unique | Specifies a positive integer that the function converts to a hexadecimal string for use in creating the temporary filename. If Unique is zero, the function uses a hexadecimal string derived from the current system time. In this case, the function uses different values until it finds a unique filename, and then it creates the file in Directory\$. If Unique is nonzero, the function appends the hexadecimal string to Prefix\$ to form the temporary filename. In this case, the function does not create the specified file, and does not test whether the filename is unique. |
| TempFile\$ | The variable that receives the temporary filename. This string consists of characters in the ANSI character set. |

Related Commands:

[GetTempPath](#) [FileDelete](#)

FileGetTime

Reads the time stamp in a file's directory entry.

Syntax: `FileGetTime(FileName$,Hours,Minutes,Seconds,Result)`

Parameters:

| | |
|-------------------|---|
| <i>FileName\$</i> | The name of the file whose time you want to read. If the file is located outside the current directory or on a different drive, include a path preceding the filename. |
| <i>Hours</i> | An integer variable that will contain the hour the file was last written (0 to 23). |
| <i>Minutes</i> | An integer variable that will contain the minute the file was last written (0 through 59). |
| <i>Seconds</i> | An integer variable that will contain the second the file was last written (0 through 59). |
| <i>Result</i> | An integer variable that indicates the outcome of the operation. If the file was found and its time read, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remark:

This command is handy when you are using PiXCL to create an install program and you need to check a file's time of creation before overwriting it with a newer version.

Example:

This program is a variation of the one used for [FileGetDate](#). It uses the [FileGetTime](#) command to check the time of the CMD.EXE file in the Windows 95 or NT system directory, and then displays a message box showing the time in a form similar to 15:24:32.

```
{Get Windows NT system directory}
  DirGetSystem(SystemDir$)

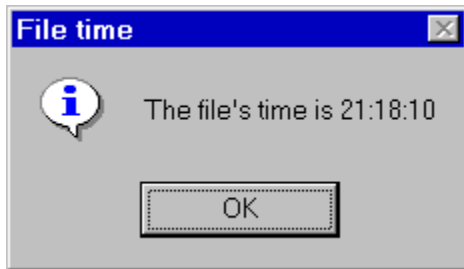
{Add \CMD.EXE to end of it}
  Cmd$ = SystemDir$ + "\CMD.EXE"

{Get CMD.EXE's time}
  FileGetTime(Cmd$,Hours,Minutes,Seconds,Result)

{Show time in message box in the form HH:MM:SS}
  If Result = 1
    TextOut$ = "The file's time is "
    Str(Hours,Hours$)
    TextOut$ = TextOut$ + Hours$
    TextOut$ = TextOut$ + ":"
    Str(Minutes,Minutes$)
    TextOut$ = TextOut$ + Minutes$
    TextOut$ = TextOut$ + ":"
    Str(Seconds,Seconds$)
    TextOut$ = TextOut$ + Seconds$
  Else
    TextOut$ = "File not found"
  Endif
Message:
```

```
MessageBox(OK,1,INFORMATION,TextOut$,"File time",Ignore)
```

The resulting MessageBox from the above script looks something like



Related Commands:

[FileGetTimeExt](#), [FileGetDate](#), [FileGetDateExt](#)

FileGetTimeExt

This function gets a file's time, returning all the time-related statistics offered by the file system.

Syntax: `FileGetTimeExt(Filename,$CREATION/LASTACCESS/LASTWRITE,
Hours,Minutes,Seconds,Milliseconds,Result)`

Parameters:

| | |
|---------------------|---|
| <i>Filename</i> \$ | The name of the file whose time you want to read. If the file is located outside the current directory, include a path preceding the filename. |
| CREATION | Statistics returned will relate to the file's time of creation. |
| LASTACCESS | Statistics returned will relate to the time the file was last accessed. |
| LASTWRITE | Statistics returned will relate to the time the file was last written. |
| <i>Hours</i> | An integer variable that will contain the file's hour setting. |
| <i>Minutes</i> | An integer variable that will contain the file's minute setting. |
| <i>Seconds</i> | An integer variable that will contain the file's seconds setting. |
| <i>Milliseconds</i> | An integer variable that will contain the file's milliseconds setting. (This setting is not offered in FAT-file systems.) |
| <i>Result</i> | An integer variable that indicates the outcome of the operation. If the file was found and its time read, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

Statistics are available for the creation time, last-access time, and last-write time. The FAT file system does not record the creation or last-access times, but both the high-performance file system (HPFS) and the New Technology file system (NTFS) do.

If the file system doesn't record a particular form of time, the statistics returned by FileGetTimeExt are unpredictable. To guard against this, use the [GetVolumeType](#) command to determine the file system used by the drive before using FileGetTimeExt.

Writing to a file changes the last write time. Writing to or reading from the file (including running an executable file) changes the last access time.

Example:

This example is a variation of the one shown for [FileGetDateExt](#). It reports the last-write, last-access, and creation dates for a chosen file. In the case of a FAT-system file, only the last-write statistics are reported.

```
{Get a filename using common dialog}
Set Filt$=
  "All Files(*.*),*.*,Write Files(*.WRI),*.wri"
FileGet(Filt$,"","","Choose a file",
  CHANGEDIR,ChosenFile$)
If ChosenFile$="" Then End
```

```

{Show LASTWRITE time info on screen}
    FileGetTimeExt (ChosenFile$, LASTWRITE,
        Hours, Minutes, Seconds, Millisecs, Result)
    DrawText (10, 10, "CREATION")
    DrawText (10, 20, "Hours")
    DrawNumber (50, 20, Hours)
    DrawText (10, 30, "Minutes")
    DrawNumber (50, 30, Minutes)
    DrawText (10, 40, "Seconds")
    DrawNumber (50, 40, Seconds)
    DrawText (10, 50, "Millisec.")
    DrawNumber (50, 50, Millisecs)

{If file system is FAT, quit drawing now}
    Substr (ChosenFile$, 3, 1, RootDir$)
    GetVolumeType (RootDir$, VolumeType$, Ignore)
    If VolumeType$ = "FAT" Then Goto Wait_for_input

{Show LASTACCESS time info on screen}
    FileGetTimeExt (ChosenFile$, LASTACCESS,
        Hours, Minutes, Seconds, Millisecs, Result)
    DrawText (90, 10, "LASTACCESS")
    DrawText (90, 20, "Hours")
    DrawNumber (130, 20, Hours)
    DrawText (90, 30, "Minutes")
    DrawNumber (130, 30, Minutes)
    DrawText (90, 40, "Seconds")
    DrawNumber (130, 40, Seconds)
    DrawText (90, 50, "Millisec.")
    DrawNumber (130, 50, Millisecs)

{Show CREATION date info on screen}
    FileGetTimeExt (ChosenFile$, CREATION,
        Hours, Minutes, Seconds, Millisecs, Result)
    DrawText (180, 10, "LASTWRITE")
    DrawText (180, 20, "Hours")
    DrawNumber (220, 20, Hours)
    DrawText (180, 30, "Minutes")
    DrawNumber (220, 30, Minutes)
    DrawText (180, 40, "Seconds")
    DrawNumber (220, 40, Seconds)
    DrawText (180, 50, "Millisec.")
    DrawNumber (220, 50, Millisecs)

Wait_for_input:
    WaitInput ()

```

Related Commands:

[FileGetTime](#), [FileGetDate](#), [FileGetDateExt](#), [GetVolumeType](#)

FileLZExpand

Uncompresses a file that was compressed with the Microsoft utility **compress.exe** with the **-r** flag specified.

Syntax: [FileLZExpand\(CompressedName\\$, RealName\\$, Result\)](#)

Parameters:

[CompressedName\\$](#) The compressed file name ending in an underscore.
[RealName\\$](#) The output file. If the file exists it is overwritten.
[Result](#) 1 if the operation was a success, otherwise 0.

Related Commands:

[FileGetExpandedName](#)

FileMove

Moves files from one directory to another. You can also move files across drives.

Syntax: `FileMove(SourceName$,DestinationName$,Result)`

Parameters:

| | |
|--------------------------------|--|
| <code>SourceName\$</code> | The name of the file you want to move. If the file is located outside the current directory or on a different drive, include a path preceding the filename. |
| <code>DestinationName\$</code> | The name of the target file. If the target file is to be located outside the current directory or on a different drive, include a path preceding the filename. |
| <code>Result</code> | An integer variable that indicates the number of files moved, or 0 if the move operation failed. |

Remarks:

PiXCL's `FileMove` command observes many of the same shortcuts as Windows move command:

`SourceName$` and `DestinationName$` can contain wildcards (? and *).

You can specify a directory (path with no filename) for `SourceName$` or `DestinationName$`. (Unlike previous versions of PiXCL, a "\" doesn't need to be present at the end of the directory name for the operation to be successful.)

* You can use a relative path (for example, "..\temp") or a fully qualified path for `SourceName$` or `DestinationName$`.

The `Result` value indicates the number of files moved. If the target filename already exists, the move command will fail, and `Result` will return 0. Hence it is always a good idea to check if the target filename already exists, and if necessary delete the old file before issuing the move command. Note that this is the same operating mode as the FileManager / Explorer.

You can use `FileMove` to move files to another drive.

Examples:

This example shows how a way to deal with a target file that already exists.

```
Source$ = "C:\BACKUP\NEW_DATA.TXT"
Dest$ = "C:\BACKUP\OLD_DATA.TXT"
FileExist(Dest$,Result)
If Result = 1 Then FileDelete(Dest$,Res)
FileMove(Source$,Dest$,Res)
```

Another example moves the file BUDGET.TXT from C:\BACKUP to D:\BACKUP:

```
FileMove("C:\BACKUP\BUDGET.TXT", "D:\BACKUP", Move_OK)
If Move_OK=0 Then Beep | End
MessageBox(OK,1,INFORMATION,"File(s) successfully moved",
           "",Temp)
```

If you change the `FileMove` command in the previous example as follows, PiXCL will move all the files in C:\BACKUP to D:\BACKUP:

```
FileMove("C:\BACKUP", "D:\BACKUP", Move_OK)
```

Related Commands:

[FileRename](#), [DirChange](#), [DiskChange](#)

FileRead_ASCII

With this command you can read arbitrary length records from anywhere in an ascii text file, such as image header metadata, or specific fields in formatted text files.

Syntax: `FileRead_ASCII(FileName$, Offset, FieldLength, Field$,Result)`

Parameters:

| | |
|--------------------|---|
| <i>FileName\$</i> | The name of the text file to be read. |
| <i>Offset</i> | Zero based positive offset from the start of the text file where <i>Field\$</i> is to be read. |
| <i>FieldLength</i> | Length of the field to be read. |
| <i>Field\$</i> | The actual data to be read. If <i>Field\$</i> does not exist it is created. If the operation fails, <i>Field\$</i> is set to a NULL string. |
| <i>Result</i> | 1 if the operation succeeds, otherwise 0. This may be because the file does not exist. |

Remarks:

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the [DirChange](#) command to reset the current disk and directory.

Related Commands:

[FileWrite_ASCII](#), [Space](#)

FileRead_ASCIIgrid

PIXCL 5 and geoPIXCL command. ASCII Grid Format: is a simple text-based format consisting of 5 lines of header information followed by rows of space delimited float point values. There can more than one space delimiter character between grid entries. Each line of **ncols** entries is terminated by a cr-lf pair. Due to the simple and relatively compact nature of this format, users who wish to import their own grid data into PiXCL and geoPiXCL should consider writing their data in this format and then importing it.

The following is a sample grid file to illustrate the manner in which an ASCII grid file can easily be constructed (do not include **comments**). The header names are lowercase.

```
ncols 10           // number of columns
nrows 10          // number of rows
xllcenter 1520000.0 //X Pos. of the center of the lower left node
yllcenter 6490000.0 //Y Pos. of the center of the lower left node
cellsize 50.0    //Cell spacing
42.7 41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2
41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2 36.6
41.3 38.3 42.7 41.4 39.7 38.6 37.8 37.2 39.9 38.1
39.9 38.1 37.8 37.2 42.7 41.4 39.7 41.3 38.3 38.6
42.7 38.6 39.9 38.1 41.4 39.7 41.3 38.3 37.8 37.2
42.7 41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2
41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2 36.6
41.3 38.3 42.7 41.4 39.7 38.6 37.8 37.2 39.9 38.1
39.9 38.1 37.8 37.2 42.7 41.4 39.7 41.3 38.3 38.6
42.7 38.6 39.9 38.1 41.4 39.7 41.3 38.3 37.8 37.2
```

Syntax: FileRead_ASCIIgrid(*GridFile\$,GridArray&[Size],Result*)

Parameters:

GridFile\$ The name of the file to be read into the array.
GridArray&[Size] The floating point array that is to be filled with the header information and grid data. *Size* is the number of elements in the array. The first five entries in the array are used to store the header values. The array has to have been previously created.
Result 1 if the array was filled, otherwise 0.

Example:

```
DirGet(SourceDir$)
GridFile2$ = SourceDir$ + "\ascii2.grid"
GridFile3$ = SourceDir$ + "\ascii3.grid"
GetASCIIgridDim(GridFile2$,nCols,nRows)
Size = nCols * nRows
Size += 5
Array(GridData&[Size],Size)
FileRead_ASCIIgrid(GridFile2$,GridData&[0],Res)
If Res = 1
    DebugMsgBox(GridData&[60] )
Else
    DebugMsgBox("Array size incorrect")
Endif
FileWrite_ASCIIgrid(GridFile3$,GridData&[0],1,Res)
```

Remarks:

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the [DirChange](#) command to reset the current disk and directory.

Related Commands:

[FileWrite_ASCIIgrid](#) [GetASCIIgridDim](#)

FileRead_Binary

With this command you can read arbitrary length records from anywhere in an binary file.

Syntax: `FileRead_Binary(FileName$,Offset,IntegerField,FWD|REV,Result)`

Parameters:

| | |
|---------------------|--|
| <i>FileName\$</i> | The name of the binary file to be read. |
| <i>Offset</i> | Zero based positive offset from the start of the binary file where <i>IntegerField</i> is to be read. |
| <i>IntegerField</i> | The actual data to be read. One 32 bit integer value is read. |
| <i>FWD REV</i> | Read bytes in forward or reverse direction into <i>IntegerField</i> . Most image file headers would use the FWD to extract accurate header values. |
| <i>Result</i> | 1 if the operation succeeds, otherwise 0. This may be because the file does not exist, or you tried to read an offset larger than the file length. |

Remarks:

This command reads a single 32 bit integer value, with bytes in forward or reverse order as they are written on the disk file i.e. REV order reads **byte0-byte1-byte2-byte3**. The integer value created is in the form **byte3-byte2-byte1-byte0**. Hence, if you read the first four bytes in a BMP file, in hex notation, **42 4d d6 e6**, the integer returned will be **e6 d6 4d 42**.

You can convert the integer to a hex string and back with the [NumToHex](#) and [HexToNum](#) commands, respectively.

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the [DirChange](#) command to reset the current disk and directory.

Related Commands:

[FileWrite_Binary](#) [HexToNum](#) [NumToHex](#)

FileRead_INI

With this command you can read any Win 3.1, Win 95/98 or Win NT system or private initialization file. The function is provided for Win 3.1 compatibility, as most new 32 bit applications use the Windows Registration Database. There are times however that you will want to create your own PiXCL application parameters, and keep them locally with the program. You can also read the INI files of other applications and extract useful information from them. For example, many applications keep a record of the window position, or the set of last files accessed. It can be very useful to have a PiXCL control or helper application access these records.

A section in an initialization file must have the following form:

```
[section]
key=string
.
.
.
```

Syntax: FileRead_INI(*FileName*\$, *Section*\$, *Key*\$, *Return*%)

Parameters:

| | |
|-------------------|---|
| <i>FileName</i> § | String that names the initialization file. If this parameter does not contain a full path to the file, Windows searches for the file in the Windows directory. |
| <i>Section</i> § | String that specifies the section containing the key name. <i>Section</i> § is not case sensitive. If this parameter is NULL, the FileRead_INI(...) command copies all section names in the file to the <i>Return</i> § string, separated by a "space". The ListBox function can be used to view or select these section names. If the section does not exist in the specified file, the <i>Return</i> § value is "No section or key of that name!". |
| <i>Key</i> § | String containing the key name whose associated string is to be retrieved. If this parameter is NULL, the FileRead_INI(...) command copies all key names in the file to the <i>Return</i> § string, separated by a "space". The ListBox function can be used to view or select these section names. If the key does not exist in the specified file, the <i>Return</i> § value is "No section or key of that name!". If the key does exist, but has no entry, the <i>Return</i> §Return\$ value is "No String". |
| <i>Return</i> § | The string that contains the returned INI file information. If <i>Return</i> § = "" (i.e. a NULL string) the INI file read operation failed. A <i>Return</i> § string cannot exceed 256 characters, and longer strings will be truncated. Generally an INI string is a few characters on one line. |

Remarks

If the string associated with *Key*§ is enclosed in single or double quotation marks, the marks are discarded when the FileRead_INI function retrieves the string.

The FileExist command should ideally be used before the FileRead_INI command, just to make sure it really exists. FileRead_INI can also be used to read the WIN.INI file.

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the DirChange command to reset the current disk and directory.

Example

The [FileRead_INI](#) command can be used to enumerate the Sections and Keys in any INI file, including those in the Windows directory. Here is a code fragment that gets an INI file, lists all sections, asks you to select a section then a key, and returns the located key string.

```
read_INI:
    DirGet(SourceDir$)
    Filter$ = "INI Files (*.ini),*.ini"
    InitFile$ = "*.ini"
    InitDir$ = SourceDir$
    Caption$ = "Select an INI file to read"
    FileGet(Filter$, InitFile$, InitDir$, Caption$,
        CHANGEDIR_EXIST,Ini_File$)
    If Ini_File$ = "" Then Goto Wait_for_input

    Section$ = ""
    Key$ = ""

    FileRead_INI(Ini_File$,Section$,Key$,Res$)
    ListBox("All Sections requested",Res$," ",Rsl$)
    If Rsl$ = "" Then Goto Wait_for_input
    Section$ = Rsl$

    FileRead_INI(Ini_File$,Section$,Key$,Res$)
    ListBox("All Keys requested",Res$," ",Rsl$)
    If Rsl$ = "" Then Goto Wait_for_input
    Key$ = Rsl$
    FileRead_INI(Ini_File$,Section$,Key$,Res$)
    Label$ = "Requested Key String for ' " + Key$
    Label$ = Label$ + "= '"
    MsgBox(OK,1,INFORMATION,Res$,Label$,Res)
    Goto Wait_for_input
```

Related Commands:

[FileWrite_INI](#), [FileExist](#)

FileRecycle

PIXCL 5 command. You can delete a file by sending it to the Recycle Bin rather than permanently deleting it from you disk.

Syntax: `FileRecycle(Filename,$,Result)`

Parameters:

Filename\$ The name of the file or files to be moved to the Recycle Bin.
Result 1 if the operation was successful, otherwise 0.

Releted Commands:

[FileDelete](#)

FileRename

Changes the name of one or more files.

Syntax: `FileRename(SourceName$,DestinationName$,Result)`

Parameters:

SourceName\$ The name of the file you want to rename. If the file is located outside the current directory or on a different drive, include a path preceding the filename.

DestinationName\$ The new name for the file. If the target file is to be located outside the current directory or on a different drive, include the path preceding the filename.

Result An integer variable that indicates the number of files renamed, or 0 if the rename operation fails.

Remarks:

The `FileRename` command is identical to the `FileMove` command in all respects. See the section on `FileMove` for additional information.

- If a file already exists under the new name in the destination location, `FileRename` will NOT overwrite it. You would need to delete the target file first.
- `FileRename` lets you rename files between paths and drives.

Example:

This example shows how a way to deal with a target file that already exists.

```
Source$ = "C:\BACKUP\NEW_DATA.TXT"
Dest$ = "C:\BACKUP\OLD_DATA.TXT"
FileExist(Dest$,Result)
If Result = 1 Then FileDelete(Dest$,Res)
FileRename(Source$,Dest$,Res)
```

This example renames the file WINNOTES.TXT in the C:\WINDOWS directory to WINNOTES.BAK:

```
FileRename("C:\WINDOWS\WINNOTES.TXT",
           "C:\WINDOWS\WINNOTES.BAK", Renamed)
If Renamed=0 Then Beep | End
MessageBox(OK,1,INFORMATION,
           "WINNOTES successfully renamed","",Temp)
```

Related Commands:

[FileMove](#), [DirChange](#), [DiskChange](#)

FileSaveAs

Gets a filename for a Save operation using the same [FileSave](#) common dialog box that appears in most Windows applications. This uses the same syntax as [FileGet](#), and is actually the same common dialog with slight changes in text. This command does NOT actually save a file, but gets the desired name for a save operation, perhaps with [FileWrite_INI](#), [FileWrite_ASCII](#) or [SaveBitmap](#).

Syntax: `FileSaveAs(Filter$,InitialFile$,InitialDir$,Caption$,
FileSaveAs_TOKEN,ChosenFile$ [, HelpTitle$, HelpMsg$])`

Parameters:

| | |
|---|--|
| Filter\$ | A string containing pairs of comma-delimited filter elements. The first element in each pair is text describing the filter (for example, "Write Files"), and the second element specifies the filter pattern (for example, "*.wri"). You can specify multiple patterns for an element by separating them with a semicolon (for example, "*.txt;*.asc;*.pxl"). If you use a null string (" ") for Filter\$, the dialog box will not display any filters. |
| InitialFile\$ | A string containing the filename you want to use to initialize the File Name edit control. If you don't want an initial file to appear, specify a null string. |
| InitialDir\$ | A string containing the initial directory you want to look at. If you want the current directory to be the initial directory, specify a null string. |
| Caption\$ | The text you want to appear in the title bar of the dialog box. If you specify a null string, PiXCL uses "Select a file" . |
| FileSaveAs_TOKEN | Determines what happens to the current directory setting on leaving the dialog box and must be one of the following tokens: |
| CHANGEDIR | On exit of the command, change the current disk\directory to the selected directory from the initial directory specified. This will affect the result of a DirGet() command. |
| CHANGEDIRMULTI | Enables file multi-select operations. |
| CHANGEDIR_EXIST | Checks if the file and path exists. An error dialog appears if the file cannot be located. |
| CHANGEDIRMULTI_EXIST | Combination of the above. |
| NOCHANGEDIR | If a new disk\directory was selected, do not change the current directory setting. |
| NOCHANGEDIRMULTI | Enables file multi-select operations. |
| NOCHANGEDIR_EXIST | Checks if the file and path exists. An error dialog appears if the file cannot be located. |
| NOCHANGEDIRMULTI_EXIST | Combination of the above. |
| Any of the above + _HLP | Displays the dialog with a Help button that displays a Messagebox with custom text and title supplied in HelpTitle\$ and HelpMsg\$. |
| PiXCL 5: Any of the above + _PROMPT e.g. | _PROMPT displays a dialog if the filename entered or selected to be saved already exists. |
| CHANGEDIR_HLP_PROMPT | |
| ChosenFile\$ | A string variable that will contain the chosen filename, including its path. If the user selects Cancel to leave the dialog, this variable is assigned a null string (""). |
| HelpTitle\$ | The title of the Help Messagebox. |

[HelpMsg\\$](#)

The help message that is displayed in the MessageBox.

Related Commands:

[DirGetSystem](#), [ListBox](#)

FileWrite_ASCII

With this command you can write arbitrary length records to anywhere in an ascii text file, such as image header metadata, or specific fields in formatted text files. You can also create 'space' filled string variables with the [Space\(\)](#) command, then write the file to disk.

Syntax: [FileWrite_ASCII](#)(*FileName\$, Offset, FieldLength, Field\$,Result*)

Parameters:

| | |
|-----------------------------|--|
| FileName\$ | The name of the text file to be written. If FileName\$ does not exist, it is created. |
| Offset | Zero based positive offset into the text file from the beginning of the file where Field\$ is to be written. |
| FieldLength | Length of the field to be written. This will usually be the same as the length of Field\$. |
| Field\$ | The actual data to be written. |
| Result | 1 if the operation succeeds, otherwise 0. |

Remarks:

If the offset into the file is before the EOF (end-of-file) marker, the [Field\\$](#) data overwrites the field in the text file. If the offset is past the EOF, the region between the EOF (which is replaced), and the offset is filled with 'space' characters. [Field\\$](#) is then appended to the file.

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the [DirChange](#) command to reset the current disk and directory.

Related Commands:

[FileRead_ASCII](#), [Space](#)

FileWrite_ASCIIgrid

PIXCL 5 and geoPIXCL command. ASCII Grid Format: is a simple text-based format consisting of 5 lines of header information followed by rows of space delimited float point values. There can more than one space delimiter character between grid entries. Each line of **ncols** entries is terminated by a cr-lf pair. Due to the simple and relatively compact nature of this format, users who wish to import their own grid data into PiXCL and geoPiXCL should consider writing their data in this format and then importing it.

The following is a sample grid file to illustrate the manner in which an ASCII grid file can easily be constructed (do not include **comments**). The header names are lowercase.

```
ncols 10           // number of columns
nrows 10          // number of rows
xllcenter 1520000.0 //X Pos. of the center of the lower left node
yllcenter 6490000.0 //Y Pos. of the center of the lower left node
cellsize 50.0    //Cell spacing
42.7 41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2
41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2 36.6
41.3 38.3 42.7 41.4 39.7 38.6 37.8 37.2 39.9 38.1
39.9 38.1 37.8 37.2 42.7 41.4 39.7 41.3 38.3 38.6
42.7 38.6 39.9 38.1 41.4 39.7 41.3 38.3 37.8 37.2
42.7 41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2
41.4 39.7 41.3 38.3 38.6 39.9 38.1 37.8 37.2 36.6
41.3 38.3 42.7 41.4 39.7 38.6 37.8 37.2 39.9 38.1
39.9 38.1 37.8 37.2 42.7 41.4 39.7 41.3 38.3 38.6
42.7 38.6 39.9 38.1 41.4 39.7 41.3 38.3 37.8 37.2
```

Syntax: FileWrite_ASCIIgrid(*GridFile\$,GridArray&[Size],DecimalPlaces,Result*)

Parameters:

GridFile\$ The name of the grid file to be create from the array. Existing files of the same name are overwritten.

GridArray&[Size] The floating point array that is to be used to write the header information and grid data. *Size* is the number of elements in the array. The first five entries in the array are used to store the header values.

DecimalPlaces The number of decimal places that grid data is to be written into *GridFile\$*.

Result 1 if the file was written, otherwise 0.

Example:

```
DirGet(SourceDir$)
GridFile2$ = SourceDir$ + "\ascii2.grid"
GridFile3$ = SourceDir$ + "\ascii3.grid"
GetASCIIgridDim(GridFile2$,nCols,nRows)
Size = nCols * nRows
Size += 5
Array(GridData&[Size],Size)
FileRead_ASCIIgrid(GridFile2$,GridData&[0],Res)
If Res = 1
    DebugMsgBox(GridData&[60] )
Else
    DebugMsgBox("Array size incorrect")
Endif
```

```
FileWrite_ASCIIgrid(GridFile3$,GridData&[0],1,Res)
```

Remarks:

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the [DirChange](#) command to reset the current disk and directory.

Related Commands:

[FileRead_ASCIIgrid](#) [GetASCIIgridDim](#)

FileWrite_Binary

With this command you can write 32 bit integer records to anywhere in a binary file, such as image headers, or specific fields in binary files.

Syntax: `FileWrite_Binary(FileName$, Offset, IntegerField, FWD|REV, Result)`

Parameters:

| | |
|------------------------------|--|
| FileName\$ | The name of the binary file to be written. If FileName\$ does not exist, it is created |
| Offset | Zero based positive byte offset into the binary file from the beginning of the file where the IntegerField is to be written. If Offset is past the EOF, the file is zero-filled past the EOF to the offset point, where IntegerField is written. |
| IntegerField | The actual data to be written. Note that IntegerField is a variable, not a static field. Using a static number will result in a syntax error. |
| FWD REV | Write bytes in forward or reverse direction from IntegerField . |
| Result | The number of bytes written if the operation succeeds, otherwise 0. The usual value is 4 (for integers) except when a new file has been created, when the value is Offset + 4. |

Remarks:

This command writes to the disk file a single 32 bit integer value, with bytes in the selected order as they are stored in the integer i.e. in reverse order **byte3-byte2-byte1-byte0**. The disk file entries written are in the form **byte0-byte1-byte2-byte3**. Hence, if the integer stored in memory is, in hex notation, **e6 d6 4d 42**, the file will be written **42 4d d6 e6**.

You can convert the integer to a hex string and back with the [NumToHex](#) and [HexToNum](#) commands, respectively.

This command is not designed to be used for large binary edit operations, as it allows one integer value to be written per call. PiXCL does not currently support integer arrays, so writing large integer files is not possible, except by zero-filling, as shown below. If binary editing is needed, there are many products available on the Internet that do this, or the MS-DOS Debug utility can be used.

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the [DirChange](#) command to reset the current disk and directory.

Examples:

It is possible to create an arbitrary sized zero filled binary file by specifying a suitable offset, as shown in this code fragment. The resulting file will be 1024 bytes long, filled with zeros, and hex value [F6B2003A](#) at offset 128.

```
MakeBinaryFile:
    Value = 0
    FileWrite_Binary(Filename$, 1020, Value, FWD, Res)
    FileGetSize(Filename$, Size)
    DrawNumber(10, 10, Size)
    HexToNum("F6B2003A", Value)
    FileWrite_Binary(Filename$, 128, Value, FWD, Res)
    Goto Wait_for_Input
```

Related Commands:

[FileRead_Binary](#) [HexToNum](#) [NumToHex](#)

FileWrite_INI

PIXCL provides you with the ability to write any system or application INI file, as well as creating or updating your own INI parameter files. A PiXCL application or helper utility for a graphics program may need to first access an INI file to set the working image directory, or window frame coordinates.

WARNING: Writing other application INI files, and especially system INI files, can cause problems if the data written is incorrect. It is quite possible to make severe enough errors to require re-installation of software. Please, be sure you fully understand the effects of your changes. The best approach is to make copy of the target INI file while you are experimenting. If you make a mistake, all you need do is copy the backup file to the target filename.

Syntax: `FileWrite_INI(INI_file$, Section$, Key$,String$,NOWARN|WARN,Return)`

Parameter:

| | |
|--------------------|---|
| <i>INI_file\$</i> | String that names the initialization file. If this parameter does not contain a full path to the file, Windows searches for the file in the Windows directory. |
| <i>Section\$</i> | String that specifies the section containing the key name. <i>Section\$</i> is not case sensitive. If the section does not exist, it is created. The name of the section is case-independent; the string can be any combination of uppercase and lowercase letters. |
| <i>Key\$</i> | String containing the name of the key to be associated with a string. If the key does not exist in the specified section, it is created. If <i>Key\$</i> is NULL, the entire section, including all entries within the section, is deleted. |
| <i>String\$</i> | The string to be written to the file. If this parameter is NULL, the key pointed to by the <i>Key\$</i> parameter is deleted. A <i>String\$</i> variable cannot exceed 256 characters. Generally an INI string is a few characters on one line. |
| <i>NOWARN WARN</i> | Disables or enables the display of a messagebox when INI data is to be deleted. See Remarks below. In most cases you will use the NOWARN token. WARN should be used when system INI files are being written. |
| <i>Result</i> | 0 if the write operation failed, otherwise is 1. |

Remarks:

If the *Key\$* or *String\$* parameter is NULL, and the **WARN** token is used, a messagebox automatically appears warning you that you are about to delete INI file data, and that system instability is possible if you are accessing the Windows INI files such as SYSTEM.INI and WIN.INI. The messagebox provides an OK button and a Cancel button. Pressing the Cancel button will abort the `FileWrite_INI` operation, and the *Result* variable is set to 0.

Example:

```
write_INI:
  DirGet(SourceDir$)
  Filter$ = "INI Files (*.ini),*.ini"
  InitFile$ = "*.ini"
  InitDir$ = SourceDir$
  Caption$ = "Select an INI file to write"
  FileGet(Filter$, InitFile$, InitDir$, Caption$,
    CHANGEDIR_EXIST,Ini_File$)
  If Ini_File$ = "" Then Goto Wait_for_input

  Section$ = "V-image"
  Key$ = "NewKey"
```

```
String$ = ""  
  
FileWrite_INI(Ini_File$,Section$,Key$,String$,NOWARN,Res)  
If Res = 0 Then Beep  
Goto Wait_for_input
```

Remarks:

Windows has the annoying habit of setting the current disk and directory when a file is accessed. If the file you are accessing is NOT in your application directory, you may need to use the [DirChange](#) command to reset the current disk and directory.

Related Commands:

[FileRead_INI](#)

FillArray

PIXCL 5 command. An array can be filled or partially filled with an initial value or string.

Syntax: `FillArray(ArrayVariable[Start_Index],Value$|Value|Value&,Filled)`

Parameters:

| | |
|---------------------------------|--|
| <i>ArrayVariable</i> | The array variable to be filled. |
| <i>Start_Index</i> | The element index from which the fill operation starts. |
| <i>Value\$ Value Value&</i> | The initialization value. It's type must match the array type. |
| <i>Filled</i> | Numeric variable . On entry, the number of elements to fill, and on exit the number of elements filled. |

Related Commands:

[CopyArray](#)

FindExecutable

Windows keeps track of file associations for a standard list of file types, and those files that you add yourself. For example, .TXT files are generally associated with the **NotePad** utility. Hence, if you double click a .TXT file, **NotePad** is automatically started by Windows. The [FindExecutable](#) command provides you with a method of identifying the associated EXE file.

Syntax: [FindExecutable](#)(*File*,\$*Path*,\$*EXE_File*,\$*Result*)

Parameters:

| | |
|-----------------------------|---|
| File \$ | The specific file for which you want the associated EXE. The full path and name is required. |
| Path \$ | The path from which the search is to be started. This can be left as NULL string if required. |
| EXE_File \$ | The full path and name of the associated EXE if one has been defined. If the file or the association does not exist, EXE_File \$ returns a NULL string. |
| Result | 1 if the command is successful, or 2 if the file cannot be found, or 3 if the Path cannot be found. |

Example:

This code fragment returns the associated EXE Files.

```
FindingEXEfiles:
    DrawBackground
    FindExecutable("f:\html\netboxes.pxl", "", EXE_File$, Res)
    DrawText(10, 10, EXE_File$)
    FindExecutable("i:\p40toolr\mmm.wav", "", EXE_File$, Res)
    DrawText(10, 35, EXE_File$)
    Goto Wait_for_Input
```

Related Commands:

[Run](#) , [RunExt](#) , [FileExist](#)

FindFile

The [FindFile](#) command provides you with a method of quickly locating any file on your system, for those times when you know the file is somewhere on a disk, but you can't recall where.

Syntax: [FindFile](#)(*File\$*,*Path\$*, *FoundFile\$*,*Result*)

Parameters:

| | |
|-----------------------------|--|
| File\$ | The specific filename, with extension. |
| Path\$ | The path from which the search is to be started. This can be left as NULL string if required. |
| FoundFile\$ | The full path and name of the file. If the file does not exist, FoundFile\$ returns a NULL string. |
| Result | 1 if the command is successful, or 2 if the file cannot be found, or 3 if the Path cannot be found. |

Example:

This code fragment returns the associated EXE Files.

```
FindingEXEfiles:
    DrawBackground
    FindFile("f:\html\netboxes.pxl","",File$,Res)
    DrawText(10,10,File$)
    Goto Wait_for_Input
```

Related Commands:

[Run](#) , [RunExt](#) , [FileExist](#) [FindExecutable](#)

FlashBMWindow

Bitmap windows are child windows within the PIXCL client area, and the created state of the title bar is inactive. If you left or right click on any bitmap window, its titlebar will be highlighted. If you want to highlight the titlebar under program control, for example, to indicate to the user that this bitmap window should be considered the active one, the [FlashBMWindow](#) command is used.

You can also use the [FlashBMWindow](#) command to set group of bitmap windows to active state.

Syntax: [FlashBMWindow\(WindowID, TOGGLE | RESET\)](#)

Parameters:

| | |
|--------------------------|--|
| WindowID | The bitmap window ID returned from a DrawBitmapWindow command. |
| TOGGLE | Change the state of the bitmap window title bar e.g. inactive becomes active. |
| RESET | Reset the state of the bitmap window title bar. |

Related Commands:

[CloseBitmapWindow](#) , [DrawBitmapWindow](#) , [SetBMWMouse](#)

Float

Convert an Integer value into a floating point point variable.

Syntax: *Float(Number,FpNumber&)*

Parameters:

Number The integer value.
FpNumber& The resulting floating point number.

Related Commands:

[Int](#) [Double](#)

For-Next Loops

PIXCL supports the structured FOR loop, as follows.

Syntax: `For` variable=*n*|variable `To` *m*|variable [`By` *p*]variable
 ... commands
 `If` <condition> `Then Break` {optional}
 ... commands
`Next`

Example:

See the sample program **ForWhile.pxl** for a detailed example.

Remarks:

PIXCL commands within a `For-Next` loop do not need to be indented, but we recommend that that you do for clarity.

If the `By` keyword and variable are omitted, the `By` value defaults to 1. Inserting a `Break` command within a `For-Next` loop will break out of the loop. If a `Break` command is located outside of a `For-Next` or `While` loop, it is ignored.

Related commands:

[If-Then](#) [While Loops](#)

FpAbs

Returns the absolute value of a floating point number.

Syntax: [FpAbs\(FpNumber&,AbsNumber&\)](#)

Parameters:

[FpNumber&](#) The floating point number to convert.
[AbsNumber&](#) The resulting absolute value.

Related Commands:

[Negate](#)

FpStr

Converts a floating point number into a string.

Syntax: `FpStr(FpNumber&,Number$)`

Parameters:

`FpNumber&` The floating point number to convert.
`Number$` The resulting string.

Related Commands:

[Float](#), [FpVal](#), [Int](#), [Str](#), [Val](#)

FpStr64

Converts a double number into a string.

Syntax: `FpStr64(FpNumber#&,Number$)`

Parameters:

FpNumber#& The double number to convert.

Number\$ The resulting string.

Related Commands:

[Float](#) [FpVal](#) [Int](#) [Str](#) [Val](#)

FpVal

Converts a floating-point string into a floating-point variable. If the string cannot be converted to a valid value, the operation fails.

Syntax: `FpVal(FpNumber$,FpNumber&,Result)`

Parameters:

| | |
|----------------------------|---|
| <code>FpNumber\$</code> | The floating point string to convert. |
| <code>FpNumber&</code> | The resulting floating point variable. |
| <code>Result</code> | 1 if the operation was successful, otherwise 0, and <code>FpNumber&</code> is set to 0.0. |

Related Commands:

[Float](#), [FpVal](#), [Int](#), [Str](#), [Val](#)

FpVal64

PIXCL 5.1 command. Converts a floating-point string into a floating-point variable. If the string cannot be converted to a valid value, the operation fails.

Syntax: `FpVal64(FpNumber$,FpNumber#&,Result)`

Parameters:

`FpNumber$` The floating point **string** to convert.
`FpNumber#&` The resulting double variable.
`Result` 1 if the operation was successful, otherwise 0, and `FpNumber#&` is set to 0.0.

Related Commands:

[Float](#) [FpVal](#) [Int](#) [Str](#) [Val](#)

FreeArrayVar FreeArrayVarAll

PIXCL 5 command. If an array is no longer required, it should be freed and the memory returned to Windows. When a PIXCL application exits, it automatically frees all array memory. All array variables can be removed from memory if needed.

Syntax: [FreeArrayVarAll](#)

Syntax: [FreeArrayVar\(ArrayVariableName\[0\]\)](#)

Parameter:

[ArrayVariableName\[0\]](#) The name of the array created by the Array command. The [0] is required, or a syntax error will be caused.

Remarks:

The command [FreeVar](#) is used to remove a string variable record from the internal list and return the memory to Windows. Using [FreeVar](#) on a string array element has no effect. You can set a string array element to an empty (" ") string.

Related Commands:

[Array](#) [FreeVar](#) [FreeVarAll](#) [FreeBitmap](#) [FreeBitmapAll](#)

FreeBitmap

Removes a bitmap (both FULL and PREVIEW modes) from memory and recovers the memory it occupied.

Syntax: `FreeBitmap(FileName$)`

Parameter:

`FileName$` The filename of the bitmap file you want to free. (The same name used with `DrawBitmap`.)

Remarks:

When you use the `DrawBitmap` command to place a bitmap in an PiXCL window, PiXCL stores the bitmap image in Windows global memory, as well as in the application memory and display contexts. These contexts are, in effect, the actual PiXCL window as you see it, one stored somewhere in global memory (the memory context), and the other on your video card memory (the display context). When your script ends, PiXCL removes the bitmap(s) from memory and recovers the space it occupies.

In most cases, we advise you to remove a bitmap from memory once you have loaded the image file. In most cases, this will not have an undesirable effects. What this approach does is free up the image file global memory, while leaving the memory and display contexts alone.

If you are retrieving the same bitmap from disk and the disk bitmap has changed from the bitmap in program memory, you **MUST** use `FreeBitMap` before you reload with `DrawBitmap`, or else the original bitmap in the memory context will be displayed.

e.g.

```
BitMap$ = "C:\images\bitmap.bmp"  
DrawBitMap(10,10,BitMap$) {line#1}  
... bitmap on disk is replaced ...  
DrawBitMap(10,10,BitMap$) {line#2}
```

will result in the original bitmap in line#1 being redrawn at line#2. The correct code is

```
BitMap$ = "C:\images\bitmap.bmp"  
DrawBitMap(10,10,BitMap$) {line#1}  
... bitmap on disk is replaced ...  
FreeBitMap(BitMap$) {extra line}  
DrawBitMap(10,10,BitMap$) {line#2}
```

The only time you might not want to use the `FreeBitmap` command is when you are drawing a bitmap repeatedly within an PiXCL window. In this case, using `FreeBitmap` will require PiXCL to retrieve the bitmap from disk again, which depending on the bitmap's size may cause a visible delay.

Example:

This example draws the 256COLOR located in the Windows directory starting at the point (5,5) in the PiXCL window. It then pauses for three seconds and removes the bitmap from memory. When you run this program, you'll notice that the bitmap does not disappear from the window when the `FreeBitmap` command is executed. Instead it remains in view until you close the PiXCL window.

```
DirGetWindows(WindowsDir$)  
Bitmap$ = WindowsDir$ + "\256color.bmp"  
DrawBitmap(5,5,Bitmap$)  
WaitInput(3000)      {Pause for 3 seconds}  
FreeBitmap(Bitmap$)  {Recover memory}
```

`WaitInput()`

Related Commands:

[FreeBitmapAll](#), [DrawBitmap](#), [DrawSizedBitmap](#), [SetColorPalette](#)

FreeBitmapAll

Removes all bitmaps from memory and reclaims the memory they occupied.

Syntax: [FreeBitmapAll](#)

Remark:

See the [FreeBitmap](#) command for the rationale behind this command.

Related Commands:

[FreeBitmap](#), [DrawBitmap](#), [DrawSizedBitmap](#)

FreeConsole

PIXCL 5 Command. Use this command to close an existing console window.

Syntax: [FreeConsole\(*Result*\)](#)

Parameter:

[Result](#) 1 if the operation was successful, otherwise 0.

Related Command

[ShowConsole](#)

FreeVar

Removes a string variable from the string variable list and recovers the memory it occupied.

Syntax: `FreeVar(String$)`

Parameter:

`String$` The name of the string variable you want to free.

Remarks:

PiXCL stores string variables in Windows global memory. If you are working with a great many string variables and memory is getting low, you may find it useful to free string variables when they are no longer needed so that the memory can be reused elsewhere. Once a PiXCL program exits, all string memory assigned during operation is freed.

Example:

This program creates a string variable and immediately frees it.

```
Street$ = "10921 Reed Hartman Highway"  
FreeVar(Street$)
```

Related Commands:

[FreeVarAll](#), [FreeBitmap](#), [FreeBitmapAll](#)

FreeVarAll

Removes all string variables from the string variable list and reclaims the memory they occupied.

Syntax: [FreeVarAll](#)

Remark:

See [FreeVar](#) for a description of how PiXCL stores string variables.

Related Commands:

[FreeVar](#), [FreeBitmap](#), [FreeBitmapAll](#)

GeoPiXCLDLLAbout

GeoPiXCL command. This command is used to display the About box from a geoPiXCL DLL as a version check.

Syntax: `GeoPiXCLDLLAbout(DLLName$,Result)`

Parameter:

DLLName\$ One of **PXLgeofn.dll** or **PXLgeofm.dll** or **PXLshape.dll** or **PXLgeotif.dll** or **PXLblobs.dll**.
Result 1 if the operation was successful, otherwise 0.

Related Commands:

None

GetASCIIgridDim

PIXCL 5 and geoPIXCL command. You can get the number of entries in an ASCII grid file, so an array of the correct size can be created.

Syntax: `GetASCIIgridDim(GridFile$,nCols,nRows)`

Parameters:

`GridFile$` The name of the file to be read into the array.
`nCols, nRows` The values read from the grid file. If an error occurs, these return 0.

Related Commands:

[FileRead_ASCIIgrid](#) [FileWrite_ASCIIgrid](#)

GetArraySize

PIXCL 5 command. Check the size of an array.

Syntax: [GetArraySize\(ArrayVariable\[0\],Size\)](#)

Parameters:

[ArrayVariable\[0\]](#) The array to check.
[Size](#) The number of elements in the array.

Related Commands:

[Array](#) [CopyArray](#) [FreeArrayVar](#) [FreeArrayVarAll](#)

GetBackground

You can retrieve the current background color, either the default color which is set when PIXCL starts, or the color set by the previous [UseBackground](#) command. The RGB values can be used as needed in other commands that specify colors.

Syntax: `GetBackground(Red,Green,Blue)`

Parameters:

Red The red value, in the range 0 - 255.
Green The green value, in the range 0 - 255.
Blue The blue value, in the range 0 - 255.

Example:

The following program fragment retrieves the current background colors, then adjusts the colors, before issuing the [Usebackground](#) and [DrawBackground](#) commands.

Backgrounds:

```
GetBackground(Red, Green, Blue)
Red += 23
Green += 44
Blue += 11
UseBackground(TRANSPARENT, Red, Green, Blue)
DrawBackground
Goto Wait_for_Input
```

Related Commands

[UseBackground](#), [ChooseColor](#), [ChooseFont](#), [DrawBackground](#) , [UsePen](#), [UseBrush](#)

GetBitMapDim

These two commands get the dimensions of a bitmap from the disk, or from an image loaded into the PIXCL image list.

Syntax: `GetBitMapDim(FileName$,Lines,Pixels,BitsPerPixel)`
`GetListBitMapDim(FileName$,Lines,Pixels,BitsPerPixel)`

Parameters:

| | |
|---------------------|---|
| <i>FileName\$</i> | The name of the bitmap to be read. You should check that the file exists before using this command. |
| <i>Lines</i> | The number of lines in the bitmap. <i>Lines</i> returns 0 if the file cannot be read or and invalid image format. |
| <i>Pixels</i> | The number of pixels per line. <i>Pixels</i> returns 0 if the file cannot be read or is an invalid format. |
| <i>BitsPerPixel</i> | The number of bits per pixel: this will be either 1, 4, 8, 24 or 32. Any other number is considered an error condition. |

Example:

The following program fragment requests a bitmap file, accesses the header, then writes the return data to the client area. Note how the **CHANGEDIR_EXIST** token is used to ensure that the file actually exists. An alternative is to use the `FileExist` command before the `GetBitMapDim` command.

```
File_Open:
  Filter$ = "All files (*.bmp),*.bmp"
  InitFile$ = "*.bmp"
  InitDir$ = SourceDir$
  Label$ = "Test of GetBitMapDim"
  FileGet(Filter$,InitFile$,InitDir$,Label$,
    CHANGEDIR_EXIST,Chosen$)
  If Chosen$ = "" Then Goto Wait_for_Input

  GetBitMapDim(Chosen$,Lines,Pixels,Bits)

  DrawBackground
  DrawNumber(10,10,Lines)
  DrawNumber(10,30,Pixels)
  DrawNumber(10,50,Bits)

  Goto Wait_for_Input
```

Remarks:

`GetListBitmapDim` sets the internal PIXCL pointer to the current bitmap, as do the `LoadBitmap` and `DrawBitmap` commands. If you load multiple bitmaps, get the dimensions of one of the loaded images, then want to, for example, perform an image processing operation of the last loaded image, you must use the `SetCurrentBitmap` command or `DrawImage` first.

Related Commands:

[DrawBitMap](#), [DrawSizedBitMap](#), [FreeBitmap](#), [FreeBitmapAll](#)

GetBitmapResolution

An image loaded into or created in the PIXCL image list can have a print resolution value in dots per inch (DPI) set. The default setting is 0, that is, no resolution information is used. This is only relevant when the image is saved to a format that supports a resolution value in the file header (e.g. BMP and TIF).

The relationship between DPI and image size is intrinsic to the term – Dots Per Inch--- (really pixels per inch). If you scan something at 400 DPI the resulting file literally has 400 dots (pixels) per linear inch of the scanned dimensions. If the dimensions of the image are 4x6 inches it means you have 4x 400 pixels by 6x400 pixels of image data. The DPI of your monitor however is =constant= at only 72 to 100 DPI . You cannot squeeze these screen pixels into a higher density to view a 400dpi image. The result has to be that the pixels in your scanned image must be spread out over a wider area... ie 4x400=1600 pixels, divided by 72 pixels per inch equals an on screen size of 22 inches at 72dpi instead of the 4 inches it would be at 400dpi.

Note that with the current version of the PXLimage library, any resolution information in a read disk bitmap (BMP or TIF) is not copied to the loaded bitmap. Use the [SetBitmapResolution](#) command to do this.

Syntax: [GetBitmapResolution\(Xdpi, Ydpi\)](#)

Parameters:

[Xdpi, Ydpi](#) The dots per inch in the X and Y axes.

Related Commands:

[SetBitmapResolution](#)

GetBMWZoom

Use this command to get the current zoom factor string from a bitmap window.

Syntax: `GetBMWZoom(WindowID,Zoom$)`

Parameters:

WindowID The ID returned from a [DrawBitmapWindow](#) command.
Zoom\$ The zoom factor returned as a string e.g. “ [\[2.50:1\]](#)“

Example:

This code fragment gets the current zoom factor string and updates the bitmap window titlebar string.

`BMWzoomer:`

```
GetBMWZoom(WinID_1,Zoom$)
Msg$ = ImageFile1$ + Zoom$
BMWinTitle(WinID_1,Msg$)
Goto Wait_for_Input
```

Related Commands:

[BMWinTitle](#) [DrawBitmapWindow](#) [ZoomBitmapWindow](#)

GetClipCursor

PIXCL 5 command. The [GetClipCursor](#) function returns the virtual screen coordinates of the rectangular area in which the cursor is active.

Syntax: [GetClipCursor\(sx1,sy1,sx2,sy2\)](#)

Parameters:

[sx1,sy1,sx2,sy2](#) The screen coordinates of the active region.

Remarks:

The cursor is a GLOBAL resource, so unless you want to leave the clip region active for all other applications, your PIXCL application must include [ClipCursor\(0,0,0,0\)](#) to reset the default.

Related Commands:

[ClipCursor](#)

GetCmdLine

It is possible to pass any number of command line arguments to any PiXCL program. These arguments are typically filenames and program control parameters.

Syntax: `GetCmdLine(CommandLine$)`

Parameters:

`CommandLine$` The complete command line including the path and the application name is returned in this string variable. e.g.
"C:\APPS\IXL_APP.EXE" arg#1 arg#2 arg#3.

The various arguments can be separated into their own string variables using the string commands available in PiXCL.

Remarks:

The path and name of the executable is returned surrounded with quotes (") while the arguments are separated by one or more space characters, as shown above. If you are intending to use a filename argument for some purpose, you will need to ensure that any leading or trailing spaces are deleted with the [Trim](#) and [StrRev](#) commands, because the command line is exactly what was passed to the EXE file: this can include multiple spaces between arguments, so you must not assume that there will be one space delimiter only.

Example:

```
GetCmdLine(CommandLine$)
Instr(CommandLine$), ".EXE",Location)
Location = Location + 4 {move to next space delimiter}
Len(CommandLine$,Long)
Long = Long - Location
Right(CommandLine$, Long, ArgList$ )
{or use the RightOf command}
Trim(ArgList$)
StrRev(ArgList$) {remove leading spaces, reverse string}
Trim(ArgList$)
StrRev(ArgList$) {remove trailing spaces, restore string}
FreeVar(CommandLine$)
```

`ArgList$` now contains the three commandline arguments without any leading or trailing spaces.

Related Commands:

All the String commands [CountCmdLineArgs](#)

GetCommPort

Gets the current baud rate, parity etc from the selected port.

Syntax: [GetCommPort\(COMx,Baud,DataBits,Parity,StopBits,XonXoff\)](#)

Parameters:

| | |
|--------------------------|--|
| COMx | Port, where x = 1 – 4 for standard PCs. PIXCL 5: If you have a multiple COM port board installed, ports 5-13 are supported. |
| Baud | The current baud rate. |
| DataBits | The current number of data bits. |
| Parity | 0 - 4 = none, odd, even, mark, space. |
| StopBits | 0 = 1 bit, 1 = 1.5 bits, 2 = 2 bits. |
| XonXoff | Data stream flow control. XOFF =0, XON = 1. |

Related Commands:

[EscCommFunction](#) [ClearCommPort](#) [ReadCommPort](#) [SetCommPort](#) [WaitCommEvent](#) [WriteCommPort](#)

GetComputerName

This command returns the current name of the computer on which the PiXCL application is running. This will be a string not longer than 15 characters, in the standard character set including letters, numbers, and the following symbols:

! @ # \$ % ^ & ' (. - _ { } ~

Syntax: [GetComputerName\(Name\\$\)](#)

Parameters:

[Name\\$](#) The returned computer name.

Related Commands:

[SetComputerName](#)

GetCopyDataMsg

To make communication with other Windows programs written in PiXCL, Visual Basic, C or C++ easier, a PiXCL program can be sent a block of ASCII data that might be a command, filename or other desired text, using the Windows WM_COPYDATA message. The [GetCopyDataMsg](#) command is used to retrieve this string.

Syntax: [GetCopyDataMsg\(Message\\$\)](#)

Parameter:

[Message\\$](#) The message string that was sent from the other application. This might be a NULL string if all the message is requesting is that the PiXCL application starts executing at the specific label. Note that the label is NOT included in the retrieved message.

Remarks:

Functionally, [GetCopyDataMsg](#) is similar to the [PXLResumeAt](#) command, in that it will jump to a Label that must exist in the receiving application. That is, the sending application must know the explicit Label name for this command to work.

Hence, the receiving application that expects to receive WM_COPYDATA messages will have at least one defined function to process the message. If a PiXCL application receives a message and there is no handler for it, it is ignored.

You may also want to send a message back to the sending window that the message was received. Since Windows uses some global shared memory for the passed message, your receiving PiXCL application needs to let Windows have the time to clear the message memory. PiXCL makes an internal copy for itself. Hence

```
GetCopyDataMsg (Message$)
SendCopyDataMsg (SendWindow$, ReturnMessage$)
```

may behave unpredictably, possibly shutting down the receiving application. It is best to use the following method.

```
GetCopyDataMsg (Message$)
WaitInput (1) {let Windows catch up}
SendCopyDataMsg (SendWindow$, ReturnMessage$)
```

So long as there is a WaitInput somewhere in the code before the [SendCopyDataMsg](#) is issued, your programs will should always work reliably.

See the [comms1.pxl](#) and [comms2.pxl](#) programs for a simple example of using [SendCopyDataMsg](#) and [GetCopyDataMsg](#) commands.

Related Commands:

[SendCopyDataMsg](#) [PXLResumeAt](#)

GetCPUInfo

Report the CPU type and number of CPUs in the system.

Syntax: *GetCPUInfo(CPUtype,NumberOfCPUs,Speed, SerialNumberLo, SerialNumberHi)*

Parameters:

| | |
|----------------------------|--|
| <i>CPUtype</i> | 386, 486 or 586 = Pentium class, including P-II, P-III and P-IV |
| <i>NumberOfCPUs</i> | At least 1. |
| <i>Speed</i> | The identified processor raw speed in MHz. This may be slightly different to the rated speed of your system. For example, a "500 MHz" system may report 497 or 498. This is normal. |
| <i>SerialNumberLo Hi</i> | If the Pentium III and later serial number reporting is enabled in BIOS, returns the serial number, otherwise both are 0. These numbers are base 10. The Intel serial number utility reports the serial number as, say, 00000681 0001A740 19D060A9 . PiXCL reports the serial number as HI: 108352 LO: 433086633 . If you convert these numbers to hexadecimal i.e. 0001A740 19D060A9 which is the same. The leading 00000681 is not related to the actual serial number, and should be ignored. |

Related Commands:

[HexToNum](#) [NumToHex](#)

GetDialogUnits

The GetDialogUnits command returns the dialog box base units used by Windows to create dialog boxes. Both Windows and applications use these units to convert the width and height of dialog boxes and controls from dialog units, as given in dialog box templates, to pixels, and vice versa.

Syntax: `GetDialogUnits(BaseUnitX, BaseUnitY)`

Parameters:

BaseUnitX The horizontal base unit is equal to the average width, in pixels, of the characters in the system font,

BaseUnitY The vertical base unit is equal to the height, in pixels, of the system font.

Furthermore, each horizontal base unit is equal to 4 horizontal dialog units; each vertical base unit is equal to 8 vertical dialog units. Therefore, to convert dialog units to pixels, an application applies the following formulas:

$$\text{pixelX} = (\text{DialogUnitX} * \text{BaseUnitX}) / 4$$

$$\text{pixelY} = (\text{DialogUnitY} * \text{BaseUnitY}) / 8$$

Similarly, to convert from pixels to dialog units, an application applies the following formulas:

$$\text{DialogUnitX} = (\text{pixelX} * 4) / \text{BaseUnitX}$$

$$\text{DialogUnitY} = (\text{pixelY} * 8) / \text{BaseUnitY}$$

The multiplication is performed before the division to avoid rounding problems if base units are not divisible by 4 or 8.

Related Commands:

PixelsToDlgUnits DlgUnitsToPixels

GetDiskSpace

This function gets the specified disk type, total space and free space in kilobytes.

Syntax: `GetDiskSpace(Disk$, Type$, TotalSpace, FreeSpace)`

Parameters:

| | |
|-------------------|---|
| <i>Disk\$</i> | The pop-up menu item to be queried. <i>Item\$</i> must be the root directory of a disk, i.e. C:\ or D:\. |
| <i>Type\$</i> | The type of disk located is returned. It is one of the following: FIXED REMOVABLE REMOTE CD-ROM RAMDISK UNKNOWN . If the disk cannot be found, <i>Type\$</i> returns a NULL string. |
| <i>TotalSpace</i> | The total disk space in kilobytes. If the disk is not found, <i>TotalSpace</i> returns 0. |
| <i>FreeSpace</i> | The available free space in kilobytes. If the disk is not found, <i>FreeSpace</i> returns 0. If <i>Type\$</i> is CD-ROM , <i>FreeSpace</i> reports a value of 0, because a CD-ROM is not a writeable device. |

Remarks:

You can format a floppy disk using a Windows Shell command, as follows.

`FormatFloppy:`

```
Cmd$ = WinDir$ + "\rundll32.exe shell32.dll,SHFormatDrive"  
Run(Cmd$)  
Goto Wait_for_Input
```

For more details on Shell functions and control panel applets, see [Using the Windows Shell functions with PiXCL](#)

Example:

The following program presents a `ListBox` with the available drives indicated. Selecting a drive will get the information, and then it is written to the client area.

`Initialize:`

```
NewCaption$ = "Disk Type, Total and Free Space"  
UseCoordinates(PIXEL)  
WinGetActive(OldCaption$)  
WinTitle(OldCaption$,NewCaption$)  
WinLocate(NewCaption$,200,100,580,310, Res)
```

```
SetMenu("Exit!",Run_Leave,  
ENDPOPOP,  
"Get Disk",Get_disk,  
ENDPOPOP)
```

`Wait_for_input:`

```
WaitInput()
```

`Get_file:`

```
Label$ = "Select a disk from the list"  
List$ = "A:\;B:\;C:\;D:\;E:\;F:\;G\  
Delimiter$ = ";"
```



```
Disk$ = "C:\"
ListBox (Label$,List$,Delimiter$,Disk$)
If Disk$ = "" Then Goto Wait_for_input

GetDiskSpace (Disk$,Type$,Total,Free)
DrawBackGround
DrawText (10,10,Type$)
DrawNumber (10,30,Total)
DrawNumber (10,50,Free)

Goto Wait_for_input
Run_Leave:
End
```

Related Commands:

[FileGetSize](#)

GetDragList

If your program includes a [DragAcceptFile\(ENABLE,label\)](#) command, you will have to use this command to retrieve the file list that is dragged and dropped into the PiXCL client area.

Syntax: [GetDragList\(List\\$\)](#)

Parameters:

[List\\$](#) The returned string variable that contains the list of files. If there are multiple files, they are "space" delimited.

Remarks:

[List\\$](#) should not be a null string, because at least one file should have been selected. It may be a NULL string if your list is created by another program, or with [DropFileServer](#). Once you have [List\\$](#), what you do with it is up to you. This might be loading an image, processing it, and then displaying the result. See the code fragment below.

You can create a draglist from Explorer, a FileGet dialog, or assemble the list programmatically. For example, selecting multiple files with a FileGet dialog returns a string with the selected path as the first entry, followed by each filename. Windows then automatically assembles the filenames into fully defined names with attached path before supplying them to the target application.

The maximum length of the drag list buffer is 16384 bytes.

Example:

```
DragAcceptFile (ENABLE,AcceptFile)
. . .
AcceptFile:
GetDragList (ImageList$)
LoadBitmap (ImageList$,PREVIEW)
DrawBitmap(10,10,ImageList$)
DragAcceptFile (DISABLE,AcceptFile)
Goto Wait_for_Input
```

Related Commands:

[DropFileServer](#) [DragAcceptFile](#)

GetEnvString

Get the current list of the current process environment variables, with a selected delimiter.

Syntax: [GetEnvString\(Delimiter\\$,EnvString\\$\)](#)

Parameters:

[Delimiter\\$](#) Character to delimit the variables. “|” is often used.

[EnvString\\$](#) The current process environment string.

Related Commands:

[GetEnvVariable](#) [SetEnvVariable](#)

GetEnvVariable

Use this command to get the current value of an environment variable.

Syntax: `GetEnvVariable(Variable$, Value$)`

Parameters:

`Variable$` The name of the environment variable.

`Value$` The current value of `Variable$`.

Related Commands:

[SetEnvVariable](#)

GetFontFace

Get the name of the current selected font for text display.

Syntax: [GetFontFace\(Face\\$\)](#)

Parameter:

[Face\\$](#) Either "System", or the face set by the last [UseFont](#) command.

Related Commands:

[AddFont](#) [ChooseFont](#) [RemoveFont](#) [UseFont](#) [UseFontExt](#)

GetICMProfile

PIXCL 5 command. The [GetICMProfile](#) command retrieves the file name of the current output color profile for the display monitor.

Syntax: [GetICMProfile\(ProfileName\\$, Result\)](#)

Parameters:

[ProfileName\\$](#) The name of the current colour profile. If a profile has not been installed, this returns an empty string.

[Result](#) 1 if the operation succeeded, otherwise 0.

Related Commands:

[InstallColorProfile](#) [UninstallColorProfile](#)

GetIPAddress

When an Internet IP address A.B.C.D (e.g. 192.72.14.1) rather than a Web page URL has to be entered, the [IPAddressBox](#) simplifies the process. The current address in the control is obtained with the [GetIPAddress](#) command.

Syntax: [GetIPAddress\(Packed,A,B,C,D\)](#)

Parameters:

Packed

The packed address.

A,B,C,D

The individual address fields. If the control does not exist, these fields return 0.

Related Command:

[IPAddressBox](#)

GetLocalTime, GetSystemTime

System time, actually Greenwich Mean Time or GMT, and Local system time, based on the time zone to which your PC is set can be accessed and if desired, reset. Both commands have the same arguments. The only difference is in the time that is returned. For example, in North American Eastern Standard Time, Local time may be 10:00 am, and System time will be 3:00 pm, because there is 5 hours difference from GMT. The values returned are all integers.

Syntax: `GetLocalTime(Year,Month,DayOfWeek,DayOfMonth,Hour,Minutes)`
`GetSystemTime(Year,Month,DayOfWeek,DayOfMonth,Hour,Minutes)`

PIXCL 5

`GetLocalTime(Year,Month,DayOfWeek,DayOfMonth,Hour,Minutes,Seconds)`
`GetSystemTime(Year,Month,DayOfWeek,DayOfMonth,Hour,Minutes,Seconds)`

Parameters:

| | |
|-------------------|---------------------------------|
| <i>Year</i> | Four digit integer, e.g. 1997 |
| <i>Month</i> | Range is 1 to 12 |
| <i>DayOfWeek</i> | Range is 0=Sunday to 6=Saturday |
| <i>DayOfMonth</i> | Range is 1 to 31 |
| <i>Hour</i> | Range 1 to 23 |
| <i>Minutes</i> | Range 0 to 59 |
| <i>Seconds</i> | Range 0 to 59 |

Related Commands:

[SetLocalTime, SetSystemTime](#) [GetTimeZone, TimeToASCII](#)

See also [Using the Windows Shell functions with PiXCL](#)

GetJPGOptions

PIXCL 5 command. When you load a JPEG image from disk, the options data in the file, if any, are also loaded and stored with the image in the PiXCL image list. These fields can be set or updated as well. If the disk image type is not JPEG, the command has no effect, and all values return null or 0.

Syntax: [GetJPGOptions\(FileName\\$, Comments\\$, ResType, XRes, YRes\)](#)

Parameters:

| | |
|----------------------------|--|
| FileName\$ | The name of a loaded file in the image list. |
| Comments\$ | The contents of the field. This may be an empty string. |
| ResType | The resolution indicator. 0 = no unit of measurement specified, 1 = dots per inch, 2 = pixels per meter. |
| XRes, YRes | The X and Y axis resolution. |

Related Commands:

[GetPNGOptions](#) [GetTIFFOptions](#) [SetJPGOptions](#) [SetPNGOptions](#) [SetTIFFOptions](#)

GetListBitmapPixel

The [GetPixel](#) command retrieves a pixel coordinate RGB value from the client area, which is not necessarily a 24-bit color mode in the video driver settings. The [GetListBitmapPixel](#) command gets pixel RGB value from a coordinate in a bitmap loaded into the PIXCL image list.

Syntax: [GetListBitmapPixel\(Handle,Line,Pixel,Index,R,G,B\)](#)

Parameters:

| | |
|-----------------------------|--|
| Handle | The handle of a loaded bitmap, as returned by SetCurrentBitmap , TuneImage and others. |
| Line, Pixel | The coordinate of the pixel to get the RGB values. |
| Index | The pixel index value for paletted bitmaps. Returns -1 for a 24-bit image pixel. |
| R,G,B | The Red, Green and Blue colour values. |

Related Commands:

[GetListBitmapPixelRegion](#) [SetListBitmapPixel](#) [GetPixel](#) [SetCurrentBitmap](#)

GetListBitmapPixelRegion

PIXCL 5 command. The [GetListBitmapPixel](#) command retrieves a pixel coordinate RGB value from the target image, while the [GetListBitmapPixelRegion](#) command gets pixel index and RGB values from a odd-numbered nxn region centered on the specified image coordinate.

Syntax: [GetListBitmapPixel\(Handle,Line,Pixel,PixelDataArray\[Size\], RegionSize\)](#)

Parameters:

| | |
|--------------------------------------|---|
| Handle | The handle of a loaded bitmap, as returned by SetCurrentBitmap , TuneImage and others. |
| Line, Pixel | The coordinate of the pixel to get the RGB values. Once the region size is calculated, Line and Pixel will be adjusted to allow for regions adjacent to the image edges. |
| PixelDataArray[Size] | An integer array containing the pixel index values for paletted bitmaps (returns -1 for a 24-bit image pixel), followed by the Red, Green and Blue values. <i>Size</i> defines the number of array entries, from which the maximum possible region is calculated. For example, a 3x3 region requires $4*3*3 = 36$ entries, and a 5x5 region requires $4*5*5 = 100$. PixelDataArray[Size] is also acceptable if you want to use floating point numbers immediately. |
| RegionSize | The size of the <i>RegionSize x RegionSize</i> region returned. <i>RegionSize</i> is generally used as an index into the array to extract or process the data. |

Remarks:

[GetListBitmapPixel\(Handle,Line,Pixel,Index,R,G,B\)](#) is equivalent to
[GetListBitmapPixelRegion\(Handle,Line,Pixel,PixelDataArray\[4\],RegionSize\)](#)

Related Commands;

[GetListBitmapPixel](#) [SetListBitmapPixel](#) [GetPixel](#) [SetCurrentBitmap](#)

GetMapMode

Get the current client area mapping mode.

Syntax: [GetMapMode\(*Mode*\)](#)

Parameter:

Mode Returns the mode value, where
1 = TEXT mode (the default)
7 = ISOTROPIC (0,0 is top left "corner")
8 = ANISOTROPIC (0,0 is bottom left "corner")

Related Commands:

[GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowExtent](#) [SetWindowOrigin](#)
[SetViewportExtent](#) [SetViewportOrigin](#)

GetMenuStatus

This function lets you determine whether an PiXCL menu item is grayed or checked.

Syntax: `GetMenuStatus(Item$,CHECKED/GRAYED,Result)`

Parameters:

| | |
|---------------|---|
| <i>Item\$</i> | The pop-up menu item to be queried. <i>Item\$</i> must exactly correspond to the name of the menu item as it was defined by the SetMenu command. If the menu item has an underlined letter, be sure to place an & in front of the letter when specifying <i>Item\$</i> . |
| CHECKED | Causes the function to query whether <i>Item\$</i> is checked. |
| GRAYED | Causes the function to query whether <i>Item\$</i> is grayed. |
| <i>Result</i> | An integer variable that will indicate the outcome of the query. It will be either of the following values based on whether you used CHECKED or GRAYED for the previous argument: 0 Menu item <i>is not</i> grayed/checked 1 Menu item <i>is</i> grayed/checked |

Example:

The following program sets up a menu with two options: Choice and Exit!. Beneath the Choice option is a pop-up menu with three items: Video, Toggle 'Video' option, and Check/Uncheck 'Video' option. Depending on the pop-up menu item you select, PiXCL will gray, enable, check, or uncheck the Video option within the pop-up menu.

```
{Set up the menu}
  SetMenu("&Choice", IGNORE,
    "&Video", Wait_for_input,
    "&Toggle 'Video' option", Enable,
    "&Check/Uncheck 'Video' option", Check,
    ENDPUPUP,
    "&Exit!", Shut_down,
    ENDPUPUP)

Wait_for_input:
  WaitInput()

Enable:
  GetMenuStatus("&Video", GRAYED, Result)
  If Result = 0 Then Goto Gray
  ChangeMenuItem("&Video", ENABLE, Temp)
  Goto Wait_for_input

Gray:
  ChangeMenuItem("&Video", GRAY, Temp)
  Goto Wait_for_input

Check:
  GetMenuStatus("&Video", CHECKED, Result)
  If Result = 1 Then Goto Uncheck
  ChangeMenuItem("&Video", CHECK, Temp)
  Goto Wait_for_input
```

```
Uncheck:  
    ChangeMenuItem("&Video",UNCHECK,Temp)  
    Goto Wait_for_input
```

```
Shut_down:  
    End
```

Related Commands:

[SetMenu](#), [ChangeMenuItem](#)

GetPixel

Get the pixel color value as a Red / Green / Blue integer triplet. This command is useful to see the value of a pixel of a displayed image. Note that the values returned are the video memory values, not the values from the image in main memory or the original file on disk.

For example, for the same image displayed on an 8-bit, 16-bit and 24-bit per pixel video card, the returned RGB values will be different.

Syntax: `GetPixel(X, Y, r, g, b, Result)`

Parameters:

| | |
|----------------------|--|
| <code>X, Y</code> | Specifies the type of X-Y co-ordinate for the pixel that you want to know the RGB value. |
| <code>r, g, b</code> | The red / green / blue value at the indicated co-ordinate, in the range from 0 - 255. |
| <code>Result</code> | If the function is successful, is set to 1, and if not is set to 0. |

Examples:

The `GetPixel` command lets you examine displayed pixel values. In the example, a bitmap is loaded, the mouse is set up and the mouse click co-ordinates are passed to the `GetPixel` command, which then gets the RGB values and writes them as text to the client area.

```
File$ = "frog16.bmp"  
DrawSizedBitmap(1,1,600,800,File$)  
SetMouse(1,1,600,800,Pixel_RGB,X,Y)  
WaitInput()  
GetPixel(X,Y,Red,Green,Blue,Result)  
DrawNumber(10,10,Red)  
DrawNumber(10,30,Green)  
DrawNumber(10,50,Blue)
```

Related Commands:

All the [SetMouse](#) commands.

GetScreenCaps

This function returns information about the screen display--for example, its height and width in pixels or the number of colors it is capable of displaying.

Syntax: `GetScreenCaps(Capacity,Result)`

Parameters:

| | |
|----------------------------------|--|
| <code>Capacity</code> | Specifies the type of screen information you want to retrieve. It must be one of the tokens below. |
| <code>HORZSIZE / VERTSIZE</code> | get the horizontal or vertical size in millimeters (METRIC mode) |
| <code>HORZRES / VERTRES</code> | get the horizontal or vertical size in pixels (PIXEL mode) |
| <code>NUMCOLORS</code> | get the number of colors in the static color map. |
| <code>BITSPIXEL</code> | The number of bits per pixel of the display. This will be one of 8, 15, 16, 24, or 32. |

For 8 bit displays, this returns a number (typically 20), and for displays with more than 256 colors, returns the value -1. For 24 bit displays, the static color map has 4096 entries.

| | |
|-----------------------------|--|
| <code>DESKTOPHORZRES</code> | For NT only. |
| <code>DESKTOPVERTRES</code> | Get the virtual horizontal or vertical size in pixels (PIXEL mode) |
| <code>NUMBRUSHES</code> | Get the number of active brushes. |
| <code>NUMPENS</code> | Get the number of active pens. |
| <code>NUMFONTS</code> | Get the number of available fonts. |
| <code>SIZEPALETTE</code> | Get the size of the current palette. |
| <code>COLORRES</code> | Get the screen color resolution. |
| <code>NUMRESERVED</code> | Get number of reserved colors. |

Result An integer variable that will contain the value of the desired item.

Examples:

The GetScreenCaps command can let you tailor an PiXCL program to the type of display device being used. For example, the following program loads a 256-color bitmap if the display driver supports 256 colors. Otherwise, it loads a less spectacular (but more broadly supported) 16-color bitmap. Without this test, loading the 256-color bitmap into a 16-color device would result in many areas of the bitmap being displayed in dull shades of gray when they should actually appear in color.

```
GetScreenCaps (NUMCOLORS,Colors)
If Colors = 256 Then File$ = "frog256.bmp" | Goto Draw
File$ = "frog16.bmp"
Draw:
DrawBitmap(1,1,File$)
WaitInput()
```

This next example determines the pixel resolution of the display driver. It then shows a message box indicating what was found, as shown below.

```
WinGetActive(Windowname$)
WinShow(Windowname$,HIDE,Result) {Hide PiXCL window}
GetScreenCaps (HORZRES,x)
```



```
GetScreenCaps (VERTRES,y)
Str(x,x$)
Str(y,y$)
Out$ = "Screen is " + x$
Out$ = Out$ + " x "
Out$ = Out$ + y$
Out$ = Out$ + " pixels"
MessageBox(OK,1,INFORMATION,Out$,"Resolution",Ignore)
```

Related Commands:

All the Draw commands, [UseCoordinates](#)

GetPNGOptions

PIXCL 5 command. When you load a PNG image from disk, the options data in the file, if any, are also loaded and stored with the image in the PiXCL image list. These fields can be set or updated as well. If the disk image type is not PNG, the command has no effect, and all values return empty strings.

Syntax: *GetPNGOptions(Filename\$, Title\$, Author\$, Copyright\$, Description\$, Software\$, Warning\$, Disclaimer\$, Source\$, Comment\$)*

Parameters:

| | |
|----------------------|---|
| <i>Filename\$</i> | The name of a loaded file in the image list. |
| <i>Title\$</i> | The contents of the field. This may be an empty string. |
| <i>Author\$</i> | The contents of the field. This may be an empty string. |
| <i>Copyright\$</i> | The contents of the field. This may be an empty string. |
| <i>Description\$</i> | The contents of the field. This may be an empty string. |
| <i>Software\$</i> | The contents of the field. This may be an empty string. |
| <i>Warning\$</i> | The contents of the field. This may be an empty string. |
| <i>Disclaimer\$</i> | The contents of the field. This may be an empty string. |
| <i>Source\$</i> | The contents of the field. This may be an empty string. |
| <i>Comment\$</i> | The contents of the field. This may be an empty string. |

Related Commands:

[GetJPGOptions](#) [GetTIFFOptions](#) [SetJPGOptions](#) [SetPNGOptions](#) [SetTIFFOptions](#)

GetScreenWorkArea

Windows 95 /98 / NT4 / 2000 provide the movable taskbar (also known as the system tray) that is usually located at the bottom of your screen. The taskbar can be moved to the top or either side of your screen, and can be set to multiple lines of application icons. Third party applications may also create taskbar like windows as well, for example the Visioneer ScanDirect™ iconbar.

The remaining screen space is called the **work area**. The [GetScreenWorkArea](#) command is provided so that you can easily maximize the area of your PiXCL or other application windows.

Syntax: [GetScreenWorkArea\(wx1,wy1,wx2,wy2\)](#)

Parameters:

[wx1,wy1,wx2,wy2](#) The screen coordinates of the available work area. These change with position of the Windows taskbar and other taskbar-like application windows.

Related Commands:

[TaskBarIcon](#)

GetSpecialFolder

This command displays a dialog box from the Windows shell, and provides the means to browse for a folder, whereas the [FileGet](#) command will return a filename. The set of Windows special folders are supported as the starting point, or you can set a specific defined directory. Once you have a new directory, most programs would then soon invoke the [DirChange](#) command.

Syntax: `GetSpecialFolder(Info$,Search_TOKEN,Start_TOKEN,Folder$)`

Parameters:

| | |
|---|--|
| Info\$ | The string that appears under the titlebar of the dialog. |
| Search_TOKEN | One of the following: |
| ALTSTARTUP | File system directory that corresponds to the user's nonlocalized Startup program group. |
| APPDATA | File system directory that serves as a common repository for application-specific data. |
| BITBUCKET | File system directory containing file objects in the user's Recycle Bin. The location of this directory is not in the registry; it is marked with the hidden and system attributes to prevent the user from moving or deleting it. |
| COMMON_ALTSTARTUP | File system directory that corresponds to the nonlocalized Startup program group for all users. |
| COMMON_DESKTOPDIRECTORY | File system directory that contains files and folders that appear on the desktop for all users. |
| COMMON_FAVORITES | File system directory that serves as a common repository for all users' favorite items. |
| COMMON_PROGRAMS | File system directory that contains the directories for the common program groups that appear on the Start menu for all users. |
| COMMON_STARTMENU | File system directory that contains the programs and folders that appear on the Start menu for all users. |
| COMMON_STARTUP | File system directory that contains the programs that appear in the Startup folder for all users. |
| CONTROLS | Virtual folder containing icons for the Control Panel applications. |
| COOKIES | File system directory that serves as a common repository for Internet cookies. |
| DESKTOP | Windows Desktop—virtual folder at the root of the namespace. |
| DESKTOPDIRECTORY | File system directory used to physically store file objects on the desktop (not to be confused with the desktop folder itself). |
| DRIVES | My Computer—virtual folder containing everything on the local computer: storage devices, printers, and Control Panel. The folder may also contain mapped network drives. |
| FAVORITES | File system directory that serves as a common repository for the user's favorite items. |
| FONTS | Virtual folder containing fonts. |
| HISTORY | File system directory that serves as a common repository for Internet history items. |
| INTERNET | Virtual folder representing the Internet. |
| INTERNET_CACHE | File system directory that serves as a common repository for temporary Internet files. |
| NETHOOD | File system directory containing objects that appear in the network neighborhood. |
| NETWORK | Network Neighborhood Folder—virtual folder representing the top level of the network hierarchy. |
| PERSONAL | File system directory that serves as a common repository for documents. |
| PRINTERS | Virtual folder containing installed printers. |
| PRINTHOOD | File system directory that serves as a common repository for printer |

| | |
|------------------|---|
| | links. |
| PROGRAMS | File system directory that contains the user's program groups (which are also file system directories). |
| RECENT | File system directory that contains the user's most recently used documents. |
| SENDTO | File system directory that contains Send To menu items. |
| STARTMENU | File system directory containing Start menu items. |
| STARTUP | File system directory that corresponds to the user's Startup program group. The system starts these programs whenever any user logs onto Windows NT or starts Windows 95. |
| TEMPLATES | File system directory that serves as a common repository for document templates. |

USERDEFINED *Folder\$* becomes the start path for the browse action. If *Folder\$* is an empty string, the search starts at the DeskTop special folder.

Start_TOKEN One of the following:
BROWSEFORCOMPUTER
BROWSEFORPRINTER
BROWSEINCLUDEFILES
BROWSEFORCOMPUTER_EDIT
BROWSEFORPRINTER_EDIT
BROWSEINCLUDEFILES_EDIT

Folder\$ A string containing the search result. If **USERDEFINED** is specified with a start string, and the cancel button is pressed, the current value is unchanged.

Example:

Here's a code fragment that searches for a folder starting at **C:\Temp**.

CreatingFile:

```
WaitInput(1)
Folder$ = "C:\Temp"
DlgTitle$ = "Looking for a folder, starting at " + Folder$
GetSpecialFolder(DlgTitle$,USERDEFINED ,BROWSEINCLUDEFILES, Folder$)
DebugMsgBox(Folder$)
Goto Wait_for_Input
```

Related Commands:

[DirChange](#) [FileGet](#) [FileSaveAs](#)

GetSysPowerStatus

The [GetSysPowerStatus](#) function retrieves the power status of your laptop PC or internal DC power supply, if fitted. The status indicates whether the system is running on AC or DC power, whether the battery is currently charging, and how much battery life currently remains. If a DC power system is not fitted, all return values are 0.

Syntax: [GetSysPowerStatus\(ACLineStatus, BatteryFlag, BatteryLifePercent, BatteryLifeTime, BatteryFullLifeTime\)](#)

Parameters:

[ACLineStatus](#)

AC power status. This parameter can be one of the following values:

| Value | Meaning |
|-------|-----------------|
| 0 | Offline |
| 1 | Online |
| 255 | Unknown status. |

All other values are reserved.

[BatteryFlag](#)

Battery charge status. This parameter can be a combination of the following values:

| Value | Meaning |
|-------|-------------------|
| 1 | High |
| 2 | Low |
| 4 | Critical |
| 8 | Charging |
| 128 | No system battery |
| 255 | Unknown status |

All other values are reserved.

[BatteryLifePercent](#)

Percentage of full battery charge remaining. This member can be a value in the range 0 to 100, or 255 if status is unknown. All other values are reserved.

[BatteryLifeTime](#)

Number of seconds of battery life remaining, or -1 if remaining seconds are unknown.

[BatteryFullLifeTime](#)

Number of seconds of battery life when at full charge, or -1 if full lifetime is unknown.

Remarks

Windows 95 is only capable of estimating [BatteryFullLifeTime](#) based on calculations on [BatteryLifeTime](#) and [BatteryLifePercent](#). Without smart battery subsystems, this value may not be accurate enough to be useful.

GetSystemMetrics

There are a number of system values, or metrics that are often of use in programs. These mostly relate to the size of various window components. The most generally useful can be returned with this command. Values are in pixels unless otherwise specified, and "CX" values are widths, "CY" values are heights.

Syntax: `GetSystemMetrics(TOKEN, Value)`

Parameters:

TOKEN

BOOTMODE

Value that specifies how the system was started:

- 0 Normal boot
- 1 Fail-safe boot
- 2 Fail-safe with network boot Fail-safe boot (also called SafeBoot) bypasses the user's startup files.

MOUSEBUTTONS

Number of buttons on mouse, or zero if no mouse is installed. This will normally be 2 for most systems. If the Microsoft Intellimouse™ or similar is installed, this will report 3 mouse buttons. Microsoft Intellisense™ has the same wheel mouse as Intellimouse, but a different driver that reports 2 buttons, even though the wheel button works properly with the [SetMidMouse](#) commands.

CXBORDER

Dimensions of a single border, in pixels.

CYBORDER

Dimensions of a single border, in pixels.

CXEDGE

Dimensions of a 3D border, in pixels. These are the

CYEDGE

counterparts of **CXBORDER** and **CYBORDER**.

CXCURSOR

Dimensions of standard cursor bitmaps, in pixels.

CYCURSOR

Dimensions of standard cursor bitmaps, in pixels.

CXFIXEDFRAME

Width and height of window frame for a window that can be resized.

CYFIXEDFRAME

CXFULLSCREEN

Width and height of the client area for a full-screen window.

CYFULLSCREEN

MENUBARSIZE

Height of single-line menu bar. Changes with small or large fonts, and the setting in the screen appearance dialog. Useful to find the size of title and menubars when positioning windows with [WinLocate\(\)](#).

SAMEDISPLAYFORMAT

For Windows 98 and 2000 only, 1 if all the display monitors have the same color format, 0 otherwise. Note that two displays can have the same bit depth, but different color formats. For example, the red, green, and blue pixels can be encoded with different numbers of bits, or those bits can be located in different places in a pixel's color value.

Value

The value for the selected system metric.

Related Commands:

GetTempPath

The [GetTempPath](#) function gets the temporary file path as follows:

1. The path specified by the TMP environment variable.
2. The path specified by the TEMP environment variable, if TMP is not defined.
3. The current directory, if both TMP and TEMP are not defined.

Syntax: [GetTempPath](#)(*TempPath*\$)

Parameter:

[TempPath](#)\$ The currently defined temporary path for the PiXCL process.

Related Commands:

[FileGetTempName](#) [GetEnvString](#) [GetEnvVariable](#) [SetEnvVariable](#)

GetTextSpacing

The [GetTextSpacing](#) command retrieves the current intercharacter spacing.

Syntax: [GetTextSpacing\(*Spacing*\)](#)

Parameter:

[Spacing](#) The current spacing between characters.

Remarks:

Text spacing can be different if required in the foreground and background. Use the [SetDrawMode](#) command before [GetTextSpacing](#) and [SetTextSpacing](#).

Releted Commands:

[DrawText](#) [DrawTextExt](#) [DrawNumber](#) [SetDrawMode](#) [SetTextSpacing](#)

GetTimeZone

Syntax: `GetTimeZone(Zone$)`

Parameters:

`Zone$` The time zone string. e.g. [Eastern Standard Time](#). In the unusual situation that time zone has not been defined, this returns "Unknown".

Remarks:

You can also display the Windows time zone selection dialog using a Shell command, as follows

```
Run("c:\windows\rundll32.exe shell32.dll,Control_RunDLL  
c:\windows\system\timedate.cpl,@0,1")
```

For more details on Shell functions and control panel applets, see [Using the Windows Shell functions with PiXCL](#)

Related Commands:

[GetLocalTime](#), [SetLocalTime](#), [TimeToASCII](#)

GetTIFOptions

PIXCL 5 command. When you load a TIF image from disk, the options data in the file, if any, are also loaded and stored with the image in the PiXCL image list. These fields can be set or updated as well. If the disk image type is not TIF, the command has no effect, and all values return an empty string.

Syntax: `GetTIFOptions(Filename,$,Artist,$,Description,$,Software,$,HostComputer,$,DocName,$)`

Parameters:

| | |
|------------------------|---|
| <i>Filename</i> \$ | The name of a loaded file in the image list. |
| <i>Artist</i> \$ | The contents of the field. This may be an empty string. |
| <i>Description</i> \$ | The contents of the field. This may be an empty string. |
| <i>Software</i> \$ | The contents of the field. This may be an empty string. |
| <i>HostComputer</i> \$ | The contents of the field. This may be an empty string. |
| <i>DocName</i> \$ | The contents of the field. This may be an empty string. |

Related Commands:

[GetJPGOptions](#) [GetPNGOptions](#) [SetJPGOptions](#) [SetPNGOptions](#) [SetTIFOptions](#)

GetToolBarBtnStatus

You can check the current status of a [ToolBar](#) or [ToolWindow](#) button in the same manner as the [GetMenuStatus](#) command does with menu items.

Syntax:

[GetToolBarBtnStatus](#)(*Name\$*, *BtnIndex*, *state_TOKEN*, *Result*)

Parameters:

Name\$ If NULL (""), refers to the current [ToolBar](#) command, if present. Otherwise, *Name\$* refers to a current [ToolWindow](#) title. If the toolbar or toolwindow is not present, the command is ignored. *Result* returns 0.

BtnIndex Button index, starting from 1, as defined in the Toolbar command in effect. [SEPARATOR](#) regions are counted as buttons.

state_TOKEN [ENABLED](#) | [CHECKED](#) | [PRESSED](#) | [DISABLED](#)

Result 1 if the button is changed to the selected [state](#), otherwise 0.

Remarks:

If you send a change state message to a [SEPARATOR](#) region, it is ignored and *Result* returns 1.

Example:

See the sample program [toolbars.pxl](#).

Related Commands:

[ChangeToolBarBtn](#) [ToolBar](#) [ToolWindow](#)

GetUserName

PIXCL 5 command. Returns the name of the currently logged on user.

Syntax: [GetUserName\(*Username*\)](#)

Parameter:

[UserName](#) Either the name of the user or often “default”

Related Commands:

[GetComputerName](#)

GetViewportExtent

Get the X and Y extent of the current viewport. The default viewport is the whole client area.

Syntax: [GetViewPortExtent\(Xextent, Yextent\)](#)

Parameters:

[Xextent](#) The viewport extent in the X-axis. This number can be negative.

[Yextent](#) The viewport extent in the Y-axis. This number can be negative.

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowExtent](#) [SetWindowOrigin](#)
[SetViewportExtent](#) [SetViewportOrigin](#)

GetViewportOrigin

Get the X and Y origin of the viewport in the client area.

Syntax: [GetViewportOrigin\(Xorigin, Yorigin\)](#)

Parameters:

[Xorigin](#) The viewport origin in the X-axis.

[Yorigin](#) The viewport origin in the Y-axis.

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportExtent](#) [SetMapMode](#) [SetWindowExtent](#) [SetWindowOrigin](#)
[SetViewportExtent](#) [SetViewportOrigin](#)

GetVirtualScreenSize

PIXCL 5 command. Windows 98 / ME /2000 / XP all support multiple display monitors. This is implemented either with additional video cards (they don't have to be from the same manufacturer) or as dual-head single cards such as the excellent Matrox G400/450/550 series. While additional monitors are usually set to the same dimensions and adjacent to each other, they don't have to be. Windows refers to this as the virtual screen, and the [GetVirtualScreenSize](#) command returns the width and height of the bounding rectangle of all the displays. If only one display is installed or enabled, the returned values are the size of the screen, in pixels.

Syntax: [GetVirtualScreenSize](#)(*Width*, *Height*)

Parameters:

Width The width of the virtual screen.
Height The height of the virtual screen.

See Also:

[The Coordinate System.](#)

Related Commands:

None

GetVolumeInfo

This command extends the [GetVolumeType](#) command and returns more information on a specific local or network disk volume.

Syntax: `GetVolumeInfo(RootDirectory$, VolumeName$, FileSystemType$, SerialNumber, Result)`

Parameters:

| | |
|-------------------------|---|
| <i>RootDirectory\$</i> | The root directory of the volume you want to check. |
| <i>VolumeName\$</i> | The name of the specific volume. This returns a null string if the volume is unnamed or the operation fails. |
| <i>FileSystemType\$</i> | A string variable that will contain the name of the file system--FAT, NTFS, or HPFS for fixed disks, and CDFS for CD-ROM disks. Returns a null string if the operation fails. |
| <i>SerialNumber</i> | The disk serial number created when the disk is formatted. Microsoft claim that this serial number can be considered to be unique. |
| <i>Result</i> | An integer variable that indicates the outcome of the operation. If the operation was successful, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. <i>Result</i> also returns 0 if the specified drive does not exist. |

Remarks:

If *RootDirectory\$* is incomplete (for example, it's in the form "C:" or "C" instead of "C:\"), [GetVolumeType](#) won't recognize it and will return a *Result* of 0. You can access network disks as well e.g. "\\NetworkName\D:\".

Disk serial numbers are a popular and simple method of ensuring that applications run on a single system. The serial number, perhaps used in conjunction with some other system parameter like cpu type (see [GetCPUInfo](#) command) or disk capacity, can be used to calculate a unique integer that is stored somewhere in the Registry. This number provides a validation check. The downside of this method is that when a user reformats or replaces his disk, the serial number changes.

Example:

```
VolumeInfo:
    WaitInput(1)
    DrawBackground
    GetVolumeInfo("d:\",Name$,VolType$,SerialNumber,Res)
    DrawText(10,40,Name$)
    DrawText(10,60,VolType$)
    DrawNumber(10,80,SerialNumber)
    Goto Wait_for_Input
```

Related Command:

[GetVolumeType](#)

GetVolumeType

Gets the file system of the named root directory.

Syntax: `GetVolumeType(RootDirectory,$,FileSystemType,$,Result)`

Parameters:

| | |
|--------------------------|---|
| <i>RootDirectory</i> \$ | The root directory of the volume you want to check. |
| <i>FileSystemType</i> \$ | A string variable that will contain the name of the file system--FAT, NTFS, or HPFS for fixed disks, and CDFS for CD-ROM disks. |
| <i>Result</i> | An integer variable that indicates the outcome of the operation. If the operation was successful, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

If *RootDirectory*\$ is incomplete (for example, it's in the form "C:" or "C" instead of "C:\"), `GetVolumeType` won't recognize it and will return a *Result* of 0. *Result* also returns 0 if the specified drive does not exist.

You can also format a floppy disk using a Windows Shell command, as follows.

```
FormatFloppy:  
    Cmd$ = WinDir$ + "\rundll32.exe shell32.dll,SHFormatDrive"  
    Run (Cmd$)  
    Goto Wait_for_Input
```

For more details on Shell functions and control panel applets, see [Using the Windows Shell functions with PiXCL](#)

Example:

This example determines the file system used by C:\ and draws it at the point (10,10) in the PiXCL window.

```
GetVolumeType("c:\",VolType$,Result)  
If Result = 0 Then Beep | End  
DrawText(10,10,VolType$)  
WaitInput()
```

Related Commands:

[FileGetTimeExt](#) [FileGetDateExt](#) [GetVolumeInfo](#)

GetWindowExtent

Get the X and Y extent of the client area.

Syntax: [GetWindowExtent\(Xextent, Yextent\)](#)

Parameters:

[Xextent](#) The extent in the X-axis. This number can be negative.

[Yextent](#) The extent in the Y-axis. This number can be negative.

Related Commands:

[GetMapMode](#) [GetWindowOrigin](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowExtent](#) [SetWindowOrigin](#)
[SetViewportExtent](#) [SetViewportOrigin](#)

GetWindowOrigin

Get the X and Y origin of the viewport in the client area.

Syntax: [GetWindowOrigin\(Xorigin, Yorigin\)](#)

Parameters:

[Xorigin](#) The window origin in the X-axis.
[Yorigin](#) The window origin in the Y-axis.

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowExtent](#) [SetWindowOrigin](#)
[SetViewportExtent](#) [SetViewportOrigin](#)

GlobalMemStatus

The current global memory status can be accessed at any time. This will report a variety of fixed and dynamic values related to physical and virtual memory. Memory usage and free memory (physical and virtual) are very dynamic, and will often change with consecutive calls to the function. A common use of this command is to check when, for example, images loaded in the image list should be deleted.

Note that if a PiXCL application grabs a large amount of memory, once the application is terminated, all this memory is returned to Windows.

Syntax: `GlobalMemStatus(MemoryLoading, PhysicalMemory, PhysicalMemAvailable, PagingBytes, FreePagingBytes, VirtualMemTotal, VirtualMemAvailable)`

Parameters:

| | |
|-----------------------------|---|
| <i>MemoryLoading</i> | The approximate percentage of memory is use. It is possible that this may reach 100, and the system will still function. Some other applications will need to be shut down for better system performance. |
| <i>PhysicalMemory</i> | The amount of physical memory installed, in bytes. |
| <i>PhysicalMemAvailable</i> | The amount of physical memory currently available for applications. |
| <i>PagingBytes</i> | The size of the paging file, in bytes. |
| <i>FreePagingBytes</i> | The number of free space in the paging file, in bytes. |
| <i>VirtualMemTotal</i> | The total of virtual memory address space available to the process, in bytes. |
| <i>VirtualMemAvailable</i> | The amount of virtual memory address space remaining to the process, in bytes. |

Example:

Note here that byte values are converted to KB by dividing by 1024.

```
MemoryStatus: {subroutine}
    GlobalMemStatus(MemLoad, PTotal, PAvail, PageBytes, FreePageBytes, VTotal, VAvail)
    PTotal = PTotal / 1024
    PAvail = PAvail / 1024
    PageBytes = PageBytes / 1024
    FreePageBytes = FreePageBytes / 1024
    VTotal = VTotal / 1024
    VAvail = VAvail / 1024
    DrawBackGround
    UseFont("Arial", 9, 17, NOBOLD, NOITALIC, NOUNDERLINE, 0, 0, 0)
    DrawText(10, 50, "MemoryLoad") DrawNumber(180, 50, MemLoad)
    DrawText(10, 70, "Physical RAM KB") DrawNumber(180, 70, PTotal)
    DrawText(10, 90, "Available RAM KB") DrawNumber(180, 90, PAvail)
    DrawText(10, 110, "Page KB") DrawNumber(180, 110, PageBytes)
    DrawText(10, 130, "Free Page KB") DrawNumber(180, 130, FreePageBytes)

    Return
```

Related Commands:

None.

Gosub

Executes a block of code as a subroutine. When the subroutine is completed--that is, a [Return](#) command is encountered--control returns to the command following the [Gosub](#).

Syntax: [Gosub Label / Return](#)

Parameter:

[Label](#) The label in the current text file where the subroutine begins.

Remarks:

When the program returns to the calling routine, the command immediately following [Gosub](#) is executed, even if it is on the same line. This allows [Gosub](#) to be placed in the middle of a multi-command [If](#) statement.

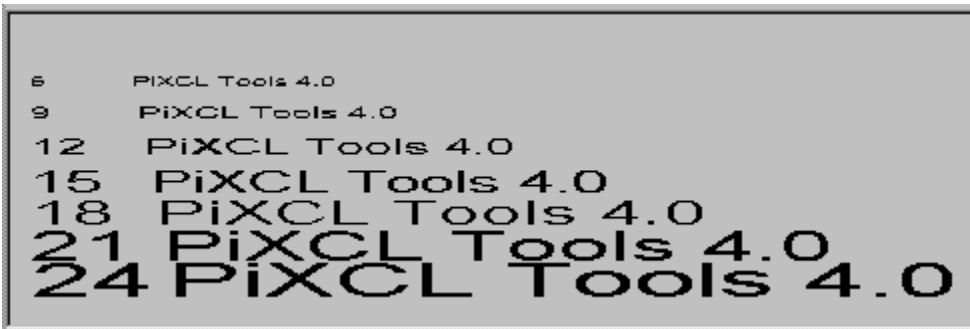
Two common programming errors:

- If the [Return](#) statement is omitted from the called subroutine, control will not return to the original program.
- Jumping out of a subroutine without executing the [Return](#).

You can use a [WaitInput](#) command within a subroutine. The application will wait for approximately the specified time, then continue executing the script. While the program is within a subroutine or nested subroutine, all other event processing for mouse, menu, toolbar and system messages is disabled. This avoids the problem of an event causing a subroutine to terminate without executing the [Return](#). Bugs of this nature are hard to track down as they don't always appear immediately.

Example:

This example uses a simple subroutine to draw text in different point sizes. The subroutine uses the Count variable to set the width of the font as well as the Y-coordinate for each line that is drawn. The figure below shows the results for an 1024x768, 256-color video driver; your results should be similar.



```
6      PIXCL Tools 4.0
9      PIXCL Tools 4.0
12     PIXCL Tools 4.0
15     PIXCL Tools 4.0
18     PIXCL Tools 4.0
21     PIXCL Tools 4.0
24     PIXCL Tools 4.0
```

Using a subroutine to draw different font sizes.

```
{Initialize counter}
    Count=6
Next:
    If Count > 25 Then Goto Wait_for_Input
    Gosub Put_Line
    Count=Count+3
    Goto Next
    Goto Wait_for_Input

Put_Line:      {Subroutine}
    Width = Count + 2
    UseFont ("Arial",Count,Width,NOBOLD,
            NOITALIC,NOUNDERLINE,0,0,0)
```

```
y=Count*5
DrawNumber(10,y,Count)
Xpos = 55 + Count
DrawText(Xpos,y,"PiXCL Tools 4.2")
Return
Wait_for_Input:
    WaitInput()
```

Related Commands:

[Goto](#), [Run](#), [RunExt](#)

Goto

Transfers control unconditionally to a label.

Syntax: *Goto Label*

Parameter:

Label A label in the current PiXCL script file.

Remark:

You can use a : (colon) following *Label* if you want. For example, PiXCL treats the following two commands identically:

```
Goto Next
Goto Next:
```

Example:

The following example launches Notepad and/or Calculator based on your responses to message boxes. The first message box asks whether you want to run Notepad and displays Yes and No buttons. If you select Yes (Button is set to 1), the program issues the Run command to launch Notepad. If you select No, the Button variable is set to 2 and the program uses a Goto to transfer control to the label Run_Calc; a similar message box then asks whether you want to run Calculator.

```
{Get PiXCL window title}
  WinGetActive(OurName$)

{Program to launch Notepad and/or Calculator}
  MessageBox(YESNO,1,QUESTION,"Run Notepad?",
             "Notepad?",Button)
  If Button=2 Then Goto Run_Calc
  Run("NOTEPAD.EXE")
  WinSetActive(OurName$,Ignore) {Bring PiXCL back to front}
Run_Calc:
  MessageBox(YESNO,1,QUESTION,"Run Calculator?",
             "Calculator?",Button)
  If Button=1 Then Run("CALC.EXE")
  End
```

Related Commands:

[Gosub If...Then If...Else...Endif](#)

GradientFillRect

A colour gradient can be generated with this command. This uses the Windows 98 supplied **msimg32.dll**. If you are running Windows 95 or NT4, you will have to get a copy of this DLL (from our Web site) and write it to the **system** directory. If the DLL is not present, this command does nothing.

Syntax: [GradientFillRect\(x1,y1,x2,y2, VERT|HORZ, sr,sg,sb, er,eg,eb\)](#)

Parameters:

| | |
|------------------------------------|---|
| x1,y1,x2,y2 | The client area coordinates of the rectangle. |
| VERT HORZ | Draw the gradient vertically or horizontally. |
| sr,sg,sb, er,eg,eb | Start end End RGB colour values. |

Related Commands:

[DrawShadeRectangle](#)

HexToNum

An eight character hexadecimal string can be converted to the equivalent positive or negative 32 bit integer. If the hex string is invalid, the conversion fails.

Syntax: `HexToNum(HexString$,Number,Result)`

Parameters:

HexString\$ An eight character string. A leading "0x" is not required.
Number The integer returned. If the conversion fails, Number returns 0.
Result 1 if the conversion succeeds, otherwise 0.

Example:

Convert:

```
Hex$ = "0034afde"  
HexToNum(Hex$,Number,Res)  
If Res = 1 Then DrawNumber(10,10,Number)
```

Related Commands:

[NumToHex](#) [Str](#) [Val](#)

Histogram

You can display multiple histograms of any loaded image or images using the [Histogram](#) command. Histograms provide you with a graphical representation of the distribution of the pixel values in the image, in either non-cumulative mode (the most commonly seen), or cumulative mode. The mean and standard deviation are also plotted on the histogram.

The [Histogram](#) command draws histogram windows as popup windows, which can float anywhere on the screen. The syntax is similar to other variable argument list commands in PiXCL, such as [ToolWindow](#), [ToolBar](#), [Button](#), [SetMenu](#) and [SetMouse](#) commands.

Histogram data is plotted in dark green, with the mean shown in red, and plus/minus one standard deviation in blue.

Syntax: [Histogram\(\)](#) Remove all histogram windows and free associated memory.

[Histogram\(x1,y1,POPUP,ImageName\\$,BitmapHandle,CUM|NONCUM, ExtraTitleData\\$,Histogram_ID\)](#)

Parameters:

| | |
|----------------------------------|---|
| x1,y1 | The top left corner coordinates of the histogram window in screen coordinates. |
| POPUP | Display mode of the histogram window. CHILD mode is not supported. |
| ImageName\$ | An image loaded into the PiXCL image list. |
| BitmapHandle | The identifier of the current bitmap for which histogram data will be calculated. |
| CUM NONCUM | Display mode of the histogram data. |
| ExtraTitleData\$ | Add extra title data with this arbitrary length string. This can be null. |
| Histogram_ID | The window identifier if the command is successful, otherwise zero. |

Remarks:

If you pass a 24 bit [BitmapHandle](#) to [Histogram](#), it will display the RED channel data only. This is normal. If you want to display the RGB channel histograms, Use the [GetChannel](#) command. Also, if the [BitmapHandle](#) is zero, or [ImageName\\$](#) has not been loaded into the list or is null, no histogram window or record is created, and [Histogram_ID](#) will return 0.

Example:

Load a single channel image and display the histogram.

[Make_Histograms:](#)

```
ImageName1$ = SourceDir$ + "\brsfbnd7.bmp"
DrawSizedBitmap(10,10,250,220,ImageName1$)
TuneImage(0,0,0,0,0,0,BitmapHandle)
Histogram(600,10,POPUP,ImageName1$,BitmapHandle,NONCUM,"Gray",HistNumber1,
          100,100,POPUP,ImageName1$,BitmapHandle,CUM,"Gray",HistNumber2)
Goto Wait_for_Input
```

Another example showing RGB channel histograms of a 24 bit image.

[GetRGBchannels:](#) {shows the HANDLE of each channel in an image.}

```
DrawSizedBitmap(10,10,250,220,ImageName1$)
GetChannel(RED,RedHandle)
GetChannel(GREEN,GreenHandle)
GetChannel(BLUE,BlueHandle)
{now create the channel histograms}
Histogram(530,100,POPUP,ImageName1$,RedHandle,NONCUM,"Red",Histogram1,
          530,260,POPUP,ImageName1$,GreenHandle,NONCUM,"Green",Histogram2,
          530,420,POPUP,ImageName1$,BlueHandle,NONCUM,"Blue",Histogram3)

Goto Wait_for_Input
```

Related Commands:

[UpdateHistogram](#) [GetChannel](#) [ReportHistogramStats](#) [ShowHistogram](#)

HTMLControl

A dialog that renders any HTML/JavaScript file or Web URL can be created with the [HTMLControl](#) command. Any HTML file or URL content that will render in Internet Explorer will render in the dialog. The title and dialog dimensions are defined in the HTML file, and the dialog is screen centered. Note that Internet Explorer 4.01 or later with Active Desktop must be installed. Internet Explorer 5 is recommended, as there are updates in the IE5 DLLs that fix bugs under NT4.

Syntax: [HTMLControl\(x,y,\(NO\)CENTER, HTMLFilename\\$, Arguments\\$, Selection\\$, Result\)](#)

Parameters:

| | |
|--------------------------------|--|
| x,y | The screen coordinates of the top left corner of the dialog for NOCENTER mode. Ignored for CENTER mode. |
| NOCENTER | Position the dialog according to x,y and the width and height values in the HTMLFilename\$. |
| CENTER | Ignore x,y and position the dialog in the center of the PiXCL application client area. |
| HTMLFilename\$ | The file or URL containing the HTML and almost always JavaScript that processes the arguments passed to the dialog controls. |
| Arguments\$ | A string with “;” delimited arguments that are used to initialize the dialog controls. |
| Selection\$ | A string selection in the dialog. This can be returned as an empty string. |
| Result | -1 or 0 if the dialog is cancelled, or 1 if an “OK” button or equivalent is pressed. |

Example:

See the sample file [HTMLdlg.pxl](#) and the associated [sample.htm](#) files.

Related Commands:

[WinHTMLHelp](#)

Hypot

Floating Point math library function. Returns the hypotenuse of a right triangle with sides X and Y.

Syntax: `Hypot(X&, Y&, Hypotenuse&)`

Parameters:

`X&, Y&` The sides of the triangle that form the right angle.
`Hypotenuse&` The size of the hypotenuse of the triangle.

Related Commands:

All the floating point math functions.

IDR_Add_Legend

To be released in a maintenance update.

This function allows the user to build or edit a legend within an IDRISI image or documentation file.

Syntax: `IDR_Add_Legend(FileName$,Result)`

Parameters:

FileName\$ The file type variable specifies image or values file.

Result 1 if the function was successful, otherwise 0.

Related Commands:

[IDR_Read_Legend](#)

IDR_CloseIdrisi

If IDRISI is running, this function can be used to close the IDRISI window.

Syntax: [IDR_CloseIdrisi](#)(*Result*)

Parameter:

Result The function returns 1 if the IDRISI window was successfully closed, and 0 if IDRISI was not closed (or not present).

Remarks:

You could also use [WinClose](#) to terminate the IDRISI window, by specifying the exact window title. The [IDR_CloseIdrisi](#) function knows what window to close. In both cases, when IDRISI receives the close message, all child windows and related Help will also be closed.

Related Commands:

[WinClose](#)

IDR_DisplayComposition

To be released in a maintenance update.

[IDR_DisplayComposition](#) is used to launch the display of either an image, a vector layer, or a map composition within the IDRISI client area. It requires a Client ID as the first parameter (see [IDR_RegisterClient](#)), and will need to indicate the type of display to produce as the second parameter using one of the DISPLAY_TYPE tokens listed below.

Syntax: `IDR_DisplayComposition(ClientId,DISPLAY_TYPE,FileName$,PalName$,Result)`

Parameters:

| | |
|----------------------------|---|
| ClientId | The Idrisi ClientID returned from IDR_RegisterClient . |
| DISPLAY_TYPE | IMAGE, VECTOR or COMP |
| FileName\$ | the name of the image, vector file or map composition to display. The file's path and extension should not be used. |
| PalName\$ | In the case of an image, PalName\$ is used to specify the name of the color palette to be used to display the image. However, in the case of a vector file, it is used to specify the name of the symbol file that should be applied, whereas for a composition, it is not used and should be set to a null string. |
| Result | 1 if the operation succeeded, otherwise 0. |

Related Commands:

None

IDR_GetDataDir

Determines the path of the working data directory that IDRISI uses for access to data and output of all results. (This path information is stored in the IDRISI.ENV file in the user's IDRISI directory.)

Syntax: [IDR_GetDataDir\(*IdrisiDataDir*\\$\)](#)

Parameter:

[IdrisiDataDir](#)\$ The current working data directory. IDRISI appends a trailing backslash (""). To make this command compatible with [DirGet](#), the trailing backslash is removed.

Related Commands:

[IDR_GetDir](#) [IDR_SetDataDir](#) [DirGet](#)

IDR_GetDir

This function determines the path of the subdirectory in which the IDRISI program modules can be found i.e. the installation directory

Syntax: [IDR_GetDir\(*IdrisiDir*\)](#)

Parameter:

[IdrisiDir](#)\$ A directory string. IDRISI appends a trailing backslash (“\”). To make this command compatible with [DirGet](#), the trailing backslash is removed.

Related Commands:

[IDR_GetDataDir](#) [DirGet](#)

IDR_GetExtensions

This function reads the Idrisi initialization file (**idrisiw.env**) and returns the names of the default extensions for image, vector, and values files.

Syntax: `IDR_GetIdrisiExtensions(Img$,ImgDoc$,Vec$,VecDoc$,Value$,ValueDoc$)`

Parameters:

| | |
|-------------------|---|
| <i>Img\$</i> | image file extension. Installation default is .img |
| <i>ImgDoc\$</i> | image document extension. Installation default is .doc |
| <i>Vec\$</i> | vector file extension. Installation default is .vec |
| <i>VecDoc\$</i> | vector document file extension. Installation default is .dvc |
| <i>Value\$</i> | value file extension. Installation default is .val |
| <i>ValueDoc\$</i> | value document extension. Installation default is .dvl |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Remarks:

Since IDRISI is a 16-bit application, file extensions are limited to three characters. If a longer extension is used, the additional characters are ignored. **idrisi.env** can also be edited with Notepad or similar.

Related Commands

[IDR_SetExtensions](#)

IDR_GetLanguage

Determines the name of the language in use with IDRISI (e.g., English, French, Spanish, etc.).

Syntax: `IDR_GetLanguage(Language$)`

Parameter:

Language\$ Returns a string e.g., English, French, Spanish.

Related Commands:

None

IDR_GetProgress

`IDR_GetProgress` gets the current values in the progress structure maintained by IDRISI for a launched module.

Syntax: `IDR_GetProgress(ClientID,ProcessID,Status,ReportType,ErrorFile$,ErrorNumber,ErrorMessage$,SubstString1$,SubstString2$,Result_1,Result_2)`

Parameters:

ClientID The ID returned by `IDR_RegisterClient`.
ProcessID An ID from `IDR_Launch` or `IDR_LaunchModule`.

Status **Integer variable.** The status field indicates whether the process is active or has terminated. Possible values are 0: ACTIVE, 1: NORMAL_TERMINATE or 2: ERROR_TERMINATE.

Most client applications do not handle IDRISI error messages. However, if you do wish to report on errors that occur in the running of IDRISI modules, or know of their character from a program perspective, you can make use of the `ErrorCode`, `ErrorFile$`, `ErrorMessage$` and the `SubstString1$` and `SubstString2$` fields below. In all cases, if you are handling errors, check the `ErrorCode` field first.

ReportType **Integer variable.** IDRISI allows several reporting types, and returns the following values.

- 1: WORKING simply indicates that the module is working. If this is the indicated report type, `Result_1` and `Result_2` are set to 0.
- 2: PERCENTDONE is the most commonly used method of reporting, and is preferred. If this type is indicated, you can access the percentage of the module's work that has been completed by examining the value in `Result_1`.
- 3: PASS_X_OF_N provides information on the pass in progress and the total number of passes the module will require. X is in `Result_1` and N is in `Result_2`.
- 4: PASS_TOTALUNKNOWN is used for cases where the report indicates the pass number in progress, but the total number of passes that will be required is unknown. In this case, only `Result_1` has a valid number.
- 5: COMPLEXPASS is the most complex of all reporting types. In this case the total number of expected passes is reported in `Result_1` and the percentage done of the current pass is reported in `Result_2`.

ErrorCode When the monitoring mode (from `IDR_LaunchModule`) is FROM_IDRISI mode, IDRISI reports on the progress of all active processes and any errors that occur. Thus there is no need to report run-time errors that occur -- IDRISI will do this for you. However, in some instances, one may wish to act upon a specific error. Furthermore, if the monitoring mode is set to FROM_CLIENT, it is the client that is responsible for acting upon and reporting the error.

As the name suggests, the `ErrorCode` field contains a numeric error code for the run-time error that has occurred, and should always be checked first before any further action is taken.

If the value is a positive integer (of 1 or greater), the error message has been generated from a non-IDRISI client process (either an independent module created by yourself, or one created by another user that you have chosen to use). In addition, the error message will need to be retrieved from an error file.

However, if the code is a negative integer, the error is a standard IDRISI run-time error, and can be found in the `ErrorMessage$` field. You do not need to retrieve this yourself.

Finally, if it is zero, it is understood that the error message has been supplied directly by the client application, and that its content can be found in the `ErrorMessage` field.

ErrorFile\$ This field contains useful information only if the `ErrorCode` is a positive number of 1 or greater. In this case, the error condition has been generated by a non-IDRISI client module that is requesting that you retrieve the appropriate message from an error file. In this context, the `ErrorFile` field contains the complete name and path of the file from which the error message can be retrieved.

If `ErrorCode` is negative or zero, the contents of the `ErrorFile$` field are meaningless and should be ignored.

IDRISI-compatible error files follow the convention of Windows ".ini" files and can be accessed using the [FileRead_INI](#) command. In IDRISI, all error files have an ".err" extension, and a main file name that corresponds to the language in use (e.g., ENGLISH.ERR, ESPANOL.ERR, etc.). In addition, they contain a single section with the same name as the language. Using the error code as a key and the language name as the section. Thus, for example, the key named "-101" in the section named "English" corresponds to a condition identified as "Disk Full". The error files included with IDRISI can be examined with a text editor to appreciate their structure; but they should not be altered.

In the case of user-defined error files, the use of an ".err" extension is recommended, but not required. Therefore you can expect that [ErrorFile\\$](#) will contain the complete name and path to that file. Also, the file will contain only a single section, using the same name as the main name of the file itself. Thus if you receive reference to a file named "MYAPP.ERR", the single section heading will read "[myapp]".

In some cases, the error message strings listed in an error file will contain references to string substitutions using character sequences of "##1##" or "##2##". In these cases, the "##1##" and "##2##" character sequences can be found in the ErrorString1 and ErrorString2 fields, respectively. Thus, for example, the IDRISI error code -1013 corresponds to the key entry "Disk File ##1## not found". The name of the file in question is thus in the ErrorString1 field. It is important to recognize, however, that the ErrorMessage field will contain the complete error message, with all substitutions already inserted. The information in the ErrorString1 and ErrorString2 fields is supplied only for cases where you need to act upon the information provided.

[ErrorMessage\\$](#) This field contains the complete error message including all string substitutions in cases where the [ErrorCode](#) was zero or negative. In these cases it can be used directly in reporting an error condition.

[SubstString1\\$](#) A substitution string to be used in the reporting of error messages (see the section on the [ErrorFile](#) field above). Replace the "##1##" character sequence in the error message retrieved with this substitution string.

[SubstString2\\$](#) A substitution string to be used in the reporting of error messages (see the section on the [ErrorFile](#) field above). Replace the "##2##" character sequence in the error message retrieved with this substitution string.

[Result_1](#) Most return values are written here. Default value is 0.

[Result_2](#) Pass N of [PASS_X_OF_N](#) is written here. Default value is 0.

Related Commands:

[IDR_SetProgress](#) [FileRead_INI](#) [FileWrite_INI](#)

IDR_IsPresent

This function determines whether IDRISI is currently running on the host system.

Syntax: [IDR_IsPresent\(*Result*\)](#)

Parameter:

[Result](#) 1 if the IDRISI window can be located, otherwise 0.

Related Commands:

[WinExist](#)

IDR_InitProgressTracking

The [IDR_InitProgressTracking](#) function is somewhat similar to an [IDR_LaunchModule](#) function in that it returns a Process ID. However, in this case it is used to notify IDRISI that a client module has been launched and that it should begin monitoring the progress of that module. The client application is then responsible for updating the progress of that module using the [IDR_SetProgress](#) function. IDRISI will then display the progress of the module in the lower-right panel of the main IDRISI window, as well as report on any error conditions that occur. See the description of [IDR_GetProgress](#) and [IDR_SetProgress](#) for details on how this done.

Syntax: [IDR_InitProgressTracking](#)(*ClientId*,*ProcName*\$,*ProcessID*)

Parameters:

| | |
|-----------------------------|--|
| ClientId | The Client ID required in order to initiate progress tracking (see IDR_RegisterClient). |
| ProcName \$ | This can be any name, but is typically a module name, of 8 characters or less, that is reported next to the IDRISI progress gauge. |
| ProcessID | The return value of this function is the Process ID that is used as the second parameter of either the IDR_SetProgress or IDR_GetProgress functions. |

Related Commands:

[IDR_GetProgress](#) [IDR_SetProgress](#)

IDR_Launch

If IDRISI is not presently running, this function can be used to launch IDRISI. The TOKEN is used to specify how it should appear once launched. If successful, a value will not be returned until IDRISI has completed the entire launch process. Only one instance of IDRISI can be running.

Syntax: [IDR_Launch\(TOKEN,Result\)](#)

Parameters:

[TOKEN](#) HIDE, NORMAL, MINIMIZE, MAXIMIZE.

[Result](#) 1 when IDRISI has successfully launched, otherwise 0.

Related Commands

[Run](#) [RunExt](#) [WinShow](#)

IDR_LaunchModule

Launches an IDRISI module using a command line to specify the operation that should be performed. Launching a module requires a Client ID (see [IDR_RegisterClient](#)). In addition, it is necessary to specify how the launched module will be monitored. In most cases, the FROM_IDRISI option will be used. In this case, the progress of the module is available to both IDRISI and the client application. IDRISI will display the progress of the module as it usually does in the lower right-hand panel of the main IDRISI window.

In addition, IDRISI will report on any error conditions or warnings that occur. However, if monitoring mode [FROM_CLIENT](#) is chosen, IDRISI does not monitor the progress, nor does it report on any error conditions -- it is expected that the client application will do this (see [IDR_GetProgress](#)).

The name of the module to run, its command line, and the title and units of the resulting operation are all specified in this command.

The syntax of the command line is identical to that used in IDRISI macro operations except that the "x" character that normally begins each command line is not included. Thus, whereas a macro file might contain a call to OVERLAY as follows:

```
overlay x 5 landsat4 landsat3 ndvi
```

the command line that would be passed in the [CmdLn\\$](#) parameter of an [IDR_LaunchModule](#) command would simply be:

```
5 landsat4 landsat3 ndvi
```

and the name of the module would be specified in the [ModName\\$](#) string variable. Note that the [OutputTitle\\$](#) and string parameters must be specified, but may be passed as null strings if you don't wish to record values for these documentation file entries.

Syntax: [IDR_LaunchModule\(ClientId,CLIENT_OPTIONS,ModName\\$,CmdLn\\$,OutputTitle\\$,OutputUnits\\$,ProcessID\)](#)

Parameters:

| | |
|--------------------------------|---|
| ClientId | The Client ID returned from IDR_RegisterClient . |
| CLIENT_OPTIONS | Monitoring mode: "FROM_CLIENT" or "FROM_IDRISI" |
| ModName\$ | A valid IDRISI module name. |
| CmdLn\$ | Module command line. |
| OutputTitle\$ | File title, if relevant, otherwise a null string (""). |
| OutputUnits\$ | Output units, if relevant, otherwise a null string (""). |
| ProcessID | The return value of the IDR_LaunchModule function is a Process ID. A Process ID is an identifier that can be used to reference a process that has been launched by a client application. This ID can then be used to monitor the progress of that process (see IDR_GetProgress). ProcessID returns 0 if the module cannot be launched. |

Related Commands:

[IDR_Launch_Run](#)

IDR_Read_DocFile

To be released in a maintenance update.

This function reads the documentation file for a specified image, and places the results into a record structure. The record structure, ImageDoc, needs to be passed by the user and will be modified by the function. Legends must be read separately, using the IDR_Read_Legend function. The return value of the function indicates whether it has been successful (1) or not (0).

Syntax: `IDR_Read_DocFile(to be decided)`

Parameters:

Related Commands:

IDR_Read_Legend

To be released in a maintenance update.

This function returns a specified legend category found within an image or values documentation file. The return value of the function indicates whether it has been successful (1) or not (0).

Syntax: `IDR_Read_Legend(Filename$,Index,LegendText$,Result)`

Parameters:

| | |
|---------------------------|---|
| <code>Filename\$</code> | Legend filename. |
| <code>Index</code> | index into the file. |
| <code>LegendText\$</code> | Legend entry read. |
| <code>Result</code> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[IDR_Add_Legend](#)

IDR_Read_Val_DocFile

To be released in a maintenance update.

This function reads the documentation file for a specified values file, and places the results into a record structure. The record structure, ValuesDoc, needs to be passed by the user and will be modified by the function. Legends must be read separately, using the [IDR_Read_Legend](#) function. The return value of the function indicates whether it has been successful (1) or not (0).

Syntax: [IDR_Read_Val_DocFile](#)(to be decided)

Parameters:

Related Commands:

IDR_Read_Vec_DocFile

To be released in a maintenance update.

This function reads the documentation file for a specified vector file, and places the results into a record structure. The record structure, VectorDoc, needs to be passed by the user and will be modified by the function. The return value of the function indicates whether it has been successful (1) or not (0).

Syntax: `IDR_Read_Vec_DocFile(to be decided)`

Parameters:

Related Commands:

IDR_RegisterClient

The main IDRISI program (**idrisiw.exe**) is actually a server program that can serve many clients at any time. As a result, it is necessary for a client application to register itself and obtain a **Client ID**. This **Client ID** is then used to identify your specific application in all operations that initiate processes within the IDRISI system (such as launching a module, or tracking the progress of a user-defined module).

[IDR_RegisterClient](#) is the first step involved in initiating or monitoring a process. This function returns a Client ID, which may then be used in functions such as [IDR_LaunchModule](#) or [IDR_SetProgress](#). Note that a return of 0 signifies that the request has been refused (most likely because there are no further Client ID's available - a maximum of 16 clients can be tracked simultaneously). Clients that have received a Client ID should free up that ID as soon as they have finished needing to communicate with IDRISI by making a call to the [IDR_UnRegisterClient](#) function.

Syntax: [IDR_RegisterClient\(Client_ID\)](#)

Parameter:

[Client_ID](#) A value in the range 0 -16.

Related Commands:

[IDR_UnRegisterClient](#) [IDR_LaunchModule](#) [IDR_SetProgress](#)

IDR_SetDataDir

Specifies the path of the working data directory that IDRISI should use for access to data and output of all results.

Syntax: [IDR_SetDataDirectory\(*IdrisiDataDir*,\\$,*Result*\)](#)

Parameters:

[IdrisiDataDir](#)\$ The new data directory. IDRISI requires a trailing backslash. If one is not present it is automatically added before the command is executed.

[Result](#) 1 if the operation succeeded, otherwise 0.

Related Commands:

[IDR_GetDataDir](#)

IDR_SetDebugMode

The `IDR_SetDebugMode` function indicates how the Idrisi API should handle internal exceptions. These should rarely, if ever, occur. However, if DEBUG mode is `ON` (the default) the presence of an exception (run-time error) will cause the program to halt with an dialog box expressing the nature of the error. On the other hand, if DEBUG mode is set to `OFF`, the module will continue to run in the presence of an exception and will not cause an error dialog to be displayed.

Syntax: `IDR_SetDebugMode(ON | OFF,PreviousMode)`

Parameters:

| | |
|---------------------------|--|
| <code>ON</code> | Turn DEBUG mode on. |
| <code>OFF</code> | Turn DEBUG mode off. |
| <code>PreviousMode</code> | 0 if the previous mode was <code>OFF</code> , 1 if the previous mode was <code>ON</code> . |

Related Commands:

None.

IDR_SetExtensions

This function sets the default extensions for image, vector, and values files. If any of the parameters are left blank, [IDR_SetExtensions](#) maintains the previous values for those parameters.

Syntax: [IDR_SetExtensions](#)(*Img\$,ImgDoc\$,Vec\$,VecDoc\$,Value\$,ValueDoc\$,Result*)

Parameters:

| | |
|----------------------------|---|
| Img\$ | image file extension. Installation default is .img |
| ImgDoc\$ | image document extension. Installation default is .doc |
| Vec\$ | vector file extension. Installation default is .vec |
| VecDoc\$ | vector document file extension. Installation default is .dvc |
| Value\$ | value file extension. Installation default is .val |
| ValueDoc\$ | value document extension. Installation default is .dvl |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[IDR_GetExtensions](#)

IDR_SetProgress

[IDR_SetProgress](#) is used to specify the progress of a client module that is being monitored by IDRISI (as established by a call to [IDR_InitProgressTracking](#)). It requires three parameters. The first is a Client ID (see [IDR_RegisterClient](#)) and the second is the handle of the process owned by that client (see [IDR_InitProgressTracking](#)).

Syntax: `IDR_SetProgress(ClientID, ProcessID, Status, ReportType, ErrorFile$, ErrorCode, ErrorMessage$, Subst1$, Subst2$, Result_1, Result_2)`

Parameters:

| | |
|-----------------------------|---|
| ClientID | The ID returned by a previous call to IDR_RegisterClient . |
| ProcessID | The ID returned by either IDR_InitProgressTracking or IDR_LaunchModule . |
| Status | The status field indicates whether the process is active or has terminated. Token values are ACTIVE , NORMAL_TERMINATE or ERROR_TERMINATE . Most client applications do not handle IDRISI error messages. However, if you do wish to report on errors that occur in the running of IDRISI modules, or know of their character from a program perspective, you can make use of the ErrorCode , ErrorFile\$, ErrorMessage\$ and the SubstString1\$ and SubstString2\$ fields below. In all cases, if you are handling errors, check the ErrorCode field first. |
| ReportType | IDRISI allows several reporting types, and accepts the following TOKEN values. WORKING simply indicates that the module is working. If this is the indicated report type, Result_1 and Result_2 are set to 0. PERCENTDONE is the most commonly used method of reporting, and is preferred. If this type is indicated, you can set the percentage of the module's work that has been completed by setting a 0 - 100 value in Result_1 . PASS_X_OF_N sets the pass in progress and the total number of passes the module will require. X is in Result_1 and N is in Result_2 . PASS_TOTALUNKNOWN is used for cases where the report sets the pass number in progress, but the total number of passes that will be required is unknown. In this case, only Result_1 requires a valid number. COMPLEXPASS is the most complex of all reporting types. In this case the total number of expected passes is set in Result_1 and the percentage done (0 - 100) of the current pass is set in Result_2 . |
| ErrorCode | Three conventions are used in the reporting of an error in an IDRISI client module. If a negative number is used, this will be understood to refer to an IDRISI standard error, which will be accessed from the ".ERR" file of the language in effect (e.g., ENGLISH.ERR, FRENCH.ERR, etc.). This is not normally used by client applications and should generally be avoided (since they may change with later versions). If a positive error code (of 1 or greater) is specified, it is understood that this represents a key in a user-defined error file (see the section on ErrorFile\$ below). If an error code of 0 is specified, it is understood that no error file is in use and that the error is being reported directly from the client application code. Specifying an error of 0 will cause IDRISI to ignore the contents of the ErrorFile\$ field, and retrieve the message directly from the ErrorMessage\$ field. Note : It is essential that the Status token be set to ERROR_TERMINATE to indicate that an error has occurred. Otherwise, the error code information will not be used! |
| ErrorFile\$ | The ErrorFile\$ field contains the complete name and path of the file from which the error message should be retrieved. If the error code is negative (signifying an IDRISI error message) or zero (signifying that a direct error message is being used), this field will be ignored by IDRISI. |

IDRISI-compatible error files follow the convention of Windows ".ini" files and can be accessed using the [FileRead_INI](#) command. In IDRISI, all error files have an ".err" extension, and a main file name that corresponds to the language in use (e.g., ENGLISH.ERR, ESPANOL.ERR, etc.). In addition, they contain a single section with the same name as the language. using the error code as a key and the language name as the section. Thus, for example, the key named "-101"

in the section named "English" corresponds to a condition identified as "Disk Full". The error files included with IDRISI can be examined with a text editor to appreciate their structure; but they should not be altered.

In the case of user-defined error files, the use of an ".err" extension is recommended, but not required. Therefore you can expect that [ErrorFile\\$](#) will contain the complete name and path to that file. Also, the file will contain only a single section, using the same name as the main name of the file itself. Thus if you receive reference to a file named "MYAPP.ERR", the single section heading will read "[myapp]".

NOTE: User defined error files are not accessible. This is caused by a bug in the Idrisi MERCUR32.DLL.

In some cases, the error message strings listed in an error file will contain references to string substitutions using character sequences of "##1##" or "##2##". In these cases, the "##1##" and "##2##" character sequences can be found in the ErrorString1 and ErrorString2 fields, respectively. Thus, for example, the IDRISI error code -1013 corresponds to the key entry "Disk File ##1## not found". The name of the file in question is thus in the ErrorString1 field. It is important to recognize, however, that the [ErrorMessage\\$](#) field will contain the complete error message, with all substitutions already inserted. The information in the [SubstString1\\$](#) and [SubstString2\\$](#) fields is supplied only for cases where you need to act upon the information provided.

[ErrorMessage\\$](#) This field contains the complete error message including all string substitutions in cases where the [ErrorCode](#) was zero or negative. In these cases it can be used directly in reporting an error condition.

[SubstString1\\$](#) A substitution string to be used in the reporting of error messages (see the section on the [ErrorFile](#) field above). Replace the "##1##" character sequence in the error message retrieved with this substitution string.

[SubstString2\\$](#) A substitution string to be used in the reporting of error messages (see the section on the [ErrorFile\\$](#) field above). Replace the "##2##" character sequence in the error message retrieved with this substitution string.

[Result_1](#) Most values are set here. If the function is successful, [Result_1](#) returns 1, other 0.

[Result_2](#) Input value depends on the [ReportType](#). Return value is 0.

Related Commands:

[IDR_GetProgress](#)

IDR_UnRegisterClient

UnRegisters a client application and returns the Client ID for use by another application. The ClientID passed as the parameter should be the same one that was received by a previous call to [IDR_RegisterClient](#).

Syntax: [IDR_UnRegisterClient\(*Client_ID*\)](#)

Parameter:

[Client_ID](#) The client ID to unregister.

Related Commands:

[IDR_RegisterClient](#)

IDR_Write_DocFile

To be released in a maintenance update.

This function writes a documentation file for an image, places the values for the documentation into the record structure, and writes the data into the file.

Syntax: [IDR_Write_DocFile](#) (to be decided)

Parameters:

Related Commands:

If...Else...Endif

This control command executes commands conditionally in a structured manner. If the condition tested is true, script execution continues until either an [Else](#) or [Endif](#) keyword is located. If the condition tested is false, execution continues on the next line following the next located [Else](#) or [Endif](#).

Syntax: `If <condition>`

```
    <true commands>
Endif
```

and

```
If <condition>
    <true commands>
Else
    <false commands>
Endif
```

Parameters:

`<commands>` One or more PiXCL commands.

| <u>Operator</u> | <u>Meaning</u> |
|-----------------|--------------------------|
| = | equal to |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |
| <> | not equal to |

Table: **Logical Operators**

Remarks:

PiXCL will issue a syntax error if you try to compare a string with a number. Before performing such a comparison, use the [Str](#) function to convert the number to a string or the [Val](#) function to convert the string to a number.

You can embed [If...Then](#) and [If-Else-Endif](#) structures in a structured [If](#), to a depth of 16 levels. For example,

```
IfThenTest: {=== subroutine ===}
    DrawBackground
    Levels++
    DrawNumber(150,30,Levels)
    If Number1 = 0
        { here's a comment with an endif }
        DrawIcon(1,1,0,0,ICON01)
        If Number2 = 0 Then DrawIcon(80,1,0,0,ICON03)
        WaitInput(1)
    Number2++
```

```

If Number2 = 2 Then Number1 = 1
If Number2 = 4 Then Number2 = 0

{here's an embedded If-Endif}

If Number1 = 0
  DrawIcon(1,1,0,0,ICON07)
  If Number = 1
    DrawIcon(10,10,0,0,ICON10)
    If Number = 1
      DrawIcon(20,20,0,0,ICON10)
      If Number =1 Then DrawIcon(10,10,60,60,ICON13)
    Endif
  Endif
Else
  DrawIcon(1,1,0,0,ICON08)
Endif

Else
  DrawIcon(1,1,0,0,ICON04)
  If Number2 = 1 Then DrawIcon(80,1,0,0,ICON06)
  If Number2 = 2 Then Number1 = 0
  Number2++

  If Number1 = 1
    DrawIcon(1,10,0,0,ICON09)
    Number = 0
  Else
    DrawIcon(1,10,0,0,ICON10)
    Number = 1
  Endif

  Endif
Return

```

More Remarks

All comments are ignored, so it is quite acceptable to have the strings “Else“ and “Endif” embedded in a comment. When the interpreter is looking for the “Else” and “Endif” keywords, it will also ignore these strings if they are part of a numeric or string variable name, or part of a token identifier. We suggest that a good programming practice is to not use variable names or token identifiers that include the strings “Else” or “Endif”.

You can have [GoSub](#) commands within [If-Endif](#) structures, so long as the subroutine does in fact return to the structure. A [GoTo](#) statement within the [If-Endif](#) is acceptable, but a [GoTo](#) that jumps outside the structure is not only poor programming practice, but may eventually cause a syntax error in code that previously runs correctly. **This is not a bug: its been intentionally left in the parser code so a Syntax Error will be flagged.** The Syntax Error occurs because PIXCL keeps a record of the embedded pointers to [Else](#) and [Endif](#) keywords, and resets them as the code is executed. A jump outside the structure leaves a valid [Else](#) | [Endif](#) pointer, and eventually all the entries are used up.

The same situation applies to using a [GoTo](#) to jump out of a [While-EndWhile](#) loop. The correct method is to use the [Break](#) command.

Related Commands:

Goto, Gosub, Str, Val Switch

If...Then

This control command executes commands conditionally. If the condition tested is true, execution continues on the **same** line, following the **Then**. If the condition tested is false, execution continues on the next line following the If; all commands on the same line as the If are ignored.

If-Then commands had their uses, for example when there is one or two functions to be executed, but the If-Else-Endif structure is more appropriate when there is a block of commands.

Syntax: `If <condition> Then <commands>`

Parameters:

<condition> A conditional expression used to compare two integers or two strings taking the form

Item1 logical_operator Item2

where *Item1* and *Item2* are both integers or both strings and *logical_operator* is one of the logical Operators in Table 15-3. For example, the following are all valid conditional expressions:

```
Red <= 255
Mouse_x1 <> Mouse_x2
File$ = "SYSTEM.INI"
Lastname$ > "Carter"
```

<commands> One or more PiXCL commands.

| <u>Operator</u> | <u>Meaning</u> |
|-----------------|--------------------------|
| = | equal to |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |
| <> | not equal to |

Table: **Logical Operators**

Remarks:

PiXCL will issue a syntax error if you try to compare a string with a number. Before performing such a comparison, use the **Str** function to convert the number to a string or the **Val** function to convert the string to a number.

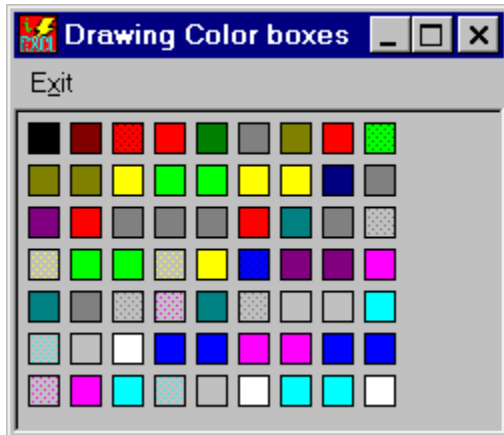
Because PiXCL treats the split vertical bar (|) as white space, it serves as a nice way to separate commands following the Then. Here's an example:

```
If x < 10 Then x = x + 1 | y=20 | Goto Next
```

Without the split vertical bars, this line would be more difficult to read because the commands following the **Then** would be all jumbled together.

Example:

This program draws an 8 by 8 matrix of colored boxes starting at (5,5). The figure below shows the positioning of the boxes in the window. Several `If` commands are used to test the value of different variables and branch accordingly. When you start the program, for example, it draws the first box in the window, increments the `Color_Red` variable by 64, and tests the value of that variable using the following `If` command:



Using `If` to test variables and draw different colored boxes based on the results

```
If Color_Red <= 255 Then Goto Color_Ready
```

In this case, if `Color_Red` is less than or equal to 255, the program branches to the `Color_Ready`: label. Otherwise, the program continues execution on the next line following the `If`.

```
{Initialize variables}
```

```
Boxes:
```

```
SetMenu("E&xit", Leave, ENDPOPUP)  
UseCaption("Drawing Color boxes")  
Color_Red=63  
Color_Green=63  
Color_Blue=63  
Color_X1=5      {Start the matrix at (5,5)}  
Color_Y1=5
```

```
Draw_Color:
```

```
UseBrush(SOLID, Color_Red, Color_Green, Color_Blue)  
Color_X2=Color_X1+16  
Color_Y2=Color_Y1+16  
DrawRectangle(Color_X1, Color_Y1, Color_X2, Color_Y2)
```

```
Color_Red=Color_Red+64  
If Color_Red<=255 Then Goto Color_Ready  
Color_Red=63  
Color_Green=Color_Green+64  
If Color_Green<=255 Then Goto Color_Ready  
Color_Green=63  
Color_Blue=Color_Blue+64
```

```
Color_Ready:
```

```
Color_X1=Color_X1+21  
If Color_X1<=184 Then Goto Draw_Color  
Color_X1=5
```

```
Color_Y1=Color_Y1+21
If Color_Y1<=144 Then Goto Draw_Color

WaitInput()
```

Remarks

See [FileGet](#), [GetScreenCaps](#), [MessageBox](#), and [Set](#) for other examples of the [If-Then](#) command. The [If-Then](#) command can be embedded in [If-Else-Endif](#) structures.

Related Commands:

[Goto](#), [Gosub](#), [Str](#), [Val](#)

ImageBox

This command creates a dialog box with an "thumbnail" or preview image display region, an 11 line text display region, and two buttons with user definable labels. The [ImageBox](#) command is designed to display a small view of an image in any of the supported formats (BMP, JPG, PCD, PCX, PPM, PSD, RAS, RLE, TGA and TIF), along with any relevant details about the image, such as the usual line and pixel counts, date information and other metadata (data about data).

Syntax: `ImageBox(Title$,ImageFile$,Text$,Btn1_Label$,Btn2_Label$,Btn)`

Parameters:

| | |
|-----------------------------|--|
| Title\$ | The string that appears in the titlebar. |
| ImageFile\$ | The image file to be displayed in the thumbnail region. |
| Text\$ | Up to eleven lines of text. |
| Btn1\$ | Label that appears on the left button. If set to "", the default label is "&Accept". |
| Btn2\$ | Label that appears on the right button. If to "", the default label is "&Reject". |
| Btn | Returned code 1 (left) or 2 (right). |

Remarks:

As with all other bitmap display related commands in PiXCL, [ImageBox](#) uses the filename as the bitmap record identifier. If the bitmap has already been loaded using the [DrawBitmap](#) or [DrawSizedBitmap](#), the bitmap is stored in memory. This memory can be recovered with the [FreeBitmap](#) or [FreeBitmapAll](#) commands. [ImageBox](#) loads Preview images into the list in the same way as the [DrawPreviewBitmap](#) command.

The bitmap is displayed in a fixed size window, and is adjusted in scale so that the largest dimension (width or height) fills the available image space. Hence, you will often get a blank area on the right side of bottom of the image area, depending on whether the bitmap is portrait, square or landscape sized.

Related Commands:

[DrawBitMap](#), [DrawPreviewBitMap](#), [DrawSizedBitmap](#), [LoadBitmap](#) , all the image processing commands.

ImageListAdd

PIXCL 5 command. Add a new image to the **ImageList** structure.

Syntax: `ImageListAdd(ImageListHandle, Image$, Mask$, ListIndex)`

Parameters:

| | |
|------------------------|---|
| <i>ImageListHandle</i> | A valid handle returned from the ImageListCreate command. |
| <i>Image\$</i> | The name of an image, either on disk or in the PiXCL image list. |
| <i>Mask\$</i> | The name of a mask image, either on disk or in the PiXCL image list. If a mask is not required set this to an empty string. |
| <i>ListIndex</i> | The returned ImageList image index, starting from 0. |

Related Commands:

[ImageListCreate](#) [ImageListDestroy](#) [ImageListDraw](#) [ImageListGetBkColor](#) [ImageListSetBkColor](#) [ImageListSetOverlay](#)

ImageListCreate

PIXCL 5 command. A part from the image list that PIXCL uses to keep track of loaded images, PIXCL also supports what is called an **ImageList** structure. This is a set of related images that are the same dimensions (lines x pixels), and provides some additional display capabilities, most notably the ability to do transparent overlays with masks.

Syntax: `ImageListCreate(Width,Height, CREATE_mode, Initial, GrowTo, ImageListHandle)`

Parameters:

Width, Height

The width and height of the images that will be in the **ImageList**.

(NO)MASK24

Create a list with 24 bit images, without or with masking.

(NO)MASK32

Create a list with 32 bit images, without or with masking.

Initial

The initial number of images in the **ImageList**. For **NOMASK** modes, this will be 1, and for **MASK** modes, 2.

GrowTo

The maximum number of images that the **ImageList** can contain.

ImageListHandle

A non-zero value that refers to the **ImageList**. The `ImageListDestroy` command is to be used when the **ImageList** is no longer required.

Related Commands:

[ImageListAdd](#) [ImageListDestroy](#) [ImageListDraw](#) [ImageListGetBkColor](#) [ImageListSetBkColor](#) [ImageListSetOverlay](#)

ImageListDestroy

PIXCL 5 command. Add a new image to the ImageList structure.

Syntax: [ImageListDestroy\(ImageListHandle, Result\)](#)

Parameters:

[ImageListHandle](#) The **ImageList** to destroy.

[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[ImageListAdd](#) [ImageListCreate](#) [ImageListDraw](#) [ImageListGetBkColor](#) [ImageListSetBkColor](#) [ImageListSetOverlay](#)

ImageListDraw

PIXCL 5 command. Draws an image from the **ImageList** structure into the client area.

Syntax: `ImageListDraw(ImageListHandle, BaseIndex, OverlayIndex, x1,y1,x2,y2, Fr, Fg, Fb, Br, Bg, Bb, MODE_token, Result)`

Parameters:

| | |
|------------------------|--|
| <i>ImageListHandle</i> | The ImageList to use, from ImageListCreate . |
| <i>BaseIndex</i> | The 0-based index of the base image to be drawn |
| <i>OverlayIndex</i> | The 1-based index of the overlay image to be drawn. |
| <i>x1,y1,x2,y2</i> | The client area draw rectangle. |
| <i>Fr,Fg,Fb</i> | Foreground colour. |
| <i>Br, Bg, Bb</i> | The background colour. |
| <i>MODE_token</i> | One of the following: BLEND25 Blend the image with 25% of the system highlight colour. BLEND50 Blend the image with 25% of the system highlight colour. MASK Blend using the mask. NORMAL Show the image as it would with the DrawBitmap command. TRANSPARENT Show <i>BaseIndex</i> overlaid with <i>OverlayIndex</i> in transparent mode. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[ImageListAdd](#) [ImageListCreate](#) [ImageListDestroy](#) [ImageListGetBkColor](#) [ImageListSetBkColor](#) [ImageListSetOverlay](#)

ImageListGetBkColor

PIXCL 5 command. Get the background **ImageList** colour values.

Syntax: [ImageListGetBkColor\(ImageListHandle, Red, Green, Blue\)](#)

Parameters:

[ImageListHandle](#) The **ImageList** to use, from [ImageListCreate](#).

[Red, Green, Blue](#) The RGB values currently used for the background colour. If these are all -1, images are drawn transparently using the mask, if one is present.

Related Commands:

[ImageListAdd](#) [ImageListCreate](#) [ImageListDestroy](#) [ImageListDraw](#) [ImageListGetBkColor](#) [ImageListSetOverlay](#)

ImageListSetBkColor

PIXCL 5 command. Set the background **ImageList** colour.

Syntax: [ImageListSetBkColor\(ImageListHandle, R,G,B, Result\)](#)

Parameters:

[ImageListHandle](#) The **ImageList** to use, from [ImageListCreate](#).

[R,G,B](#) The values to use for the background (transparency) colour. Set these all to -1 if you want to have a mask image drawn transparently.

[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[ImageListAdd](#) [ImageListCreate](#) [ImageListDestroy](#) [ImageListDraw](#) [ImageListGetBkColor](#) [ImageListSetOverlay](#)

ImageListSetOverlay

PIXCL 5 command. Sets a loaded image as the transparent overlay.

Syntax: [ImageListSetOverlay\(ImageListHandle, OverlayIndex, Result\)](#)

Parameters:

[ImageListHandle](#) The **ImageList** to use, from [ImageListCreate](#).
[OverlayIndex](#) The **ImageList** index of the overlay image.
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[ImageListAdd](#) [ImageListCreate](#) [ImageListDestroy](#) [ImageListDraw](#) [ImageListGetBkColor](#) [ImageListSetBkColor](#)

InetAutodial

This command attempts to dial into the Internet using the modem currently configured in Windows.

Syntax: [InetAutodial\(USERONLINE|UNATTENDED,Result\)](#)

Parameters:

| | |
|----------------------------|---|
| USERONLINE | Attempt a connection with user input available. |
| UNATTENDED | Attempt a connection automatically. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[InetAutodialHangup](#) [InetDial](#) [InetHangup](#)

InetAutodialHangup

This command hangs up the modem currently configured in Windows.

Syntax: [InetAutodialHangup\(*Result*\)](#)

Parameters:

[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[InetAutodial](#) [InetDial](#) [InetHangup](#)

InetCheckConnection

It is possible to check if a connection to the Internet is available before dialing up or connection on a network.

Syntax: [InetCheckConnection\(URL\\$,Result\)](#)

Parameters:

[URL\\$](#) The URL that you want to connect to directly. This can be a null string if desired.
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[InetAutodialHangup](#) [InetDial](#) [InetHangup](#)

InetDial

This command attempts to dial into the Internet using the modem currently configured in Windows.

Syntax: [InetDial\(USERONLINE|UNATTENDED,ConnectionName\\$,Connection\)](#)

Parameters:

| | |
|----------------------------------|--|
| USERONLINE | Attempt a connection with user input available. |
| UNATTENDED | Attempt a connection automatically. |
| ConnectionName\$ | A name of a dialup connection that has been configured into Windows. |
| Connection | a non-zero number if the operation was successful, otherwise 0. This number is used in the InetHangup command. |

Related Commands:

[InetAutodialHangup](#) [InetDial](#) [InetHangup](#)

InetGetConnectedState

Use this command to get the current available Internet connection method.

Syntax: [InetGetConnectedState\(State\)](#)

Parameter:

[State](#) The currently available Internet access method. Possible values are

- 1 = MODEM A modem is available
- 2 = LAN A Lan connection is available.
- 4 = PROXY Connect via a proxy.
- 8 = MODEM_BUSY The modem is assigned to another dialup application.

Related Commands:

[InetSetConnectedState](#)

InetGoOnline

This command provides a means to connect to a URL directly, using the currently defined browser.

Syntax: [InetGoOnline\(URL\\$,Result\)](#)

Parameters:

[URL\\$](#) The URL that you want to connect to directly.
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[InetAutodialHangup](#) [InetDial](#) [InetHangup](#)

InetHangup

This command hangs up the modem currently configured in Windows.

Syntax: [InetHangup\(Connection, Result\)](#)

Parameters:

[Connection](#) The connection returned by the [InetDial](#) command.
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[InetAutodial](#) [InetDial](#) [InetAutodialHangup](#)

InetSetConnectedState

This command has the effect of hanging up the modem regardless of the current state, and would generally be used to clear the modem of any previous operating modes (e.g. fax mode).

Syntax: [InetSetConnectedState\(*Result*\)](#)

Parameters:

[Result](#) 1 if the operation is successful, otherwise 0.

Related Commands:

[InetAutodialHangup](#) [InetDial](#) [InetHangup](#)

InfoMenu

Controls whether PiXCL's Info menu appears in the menu bar when creating your own custom menu.

Syntax: [InfoMenu\(REMOVE/ADD\)](#)

Parameters:

REMOVE Prevents PiXCL from including its Info menu in the menu bar the next time a [SetMenu](#) command is executed.

ADD Causes PiXCL to include the Info menu the next time a [SetMenu](#) command is executed. This is the default.

Remark:

PIXCL normally includes an Info menu as the last item in the menu bar. By using [InfoMenu\(REMOVE\)](#) before a [SetMenu](#) command, you can prevent the Info menu from appearing when creating your own custom menu.

Be sure to use [InfoMenu\(REMOVE\)](#) *before* [SetMenu](#). Using it afterward won't have any effect.

To remove the menu bar altogether, use the following code.

```
InfoMenu(REMOVE)
WaitInput(100) { Lets windows catch up}
SetMenu()
```

Removing the menu bar is useful if you want to create an image in a window, or simulate a dialog box.

Related Commands

[SetMenu](#), [Building Runtime executables](#), [AboutPiXCL](#), [AboutUser](#)

InsertListImageRect

Images loaded in the PiXCL image list can be copied into another larger list bitmap, for example when a set of thumbnail images is to be saved, or to create an animation sequence. Source and destination images must have the same number of bits per pixel.

Syntax: `InsertListImageRect(SourceImage $,DestImage$,x1,y1,x2,y2,Result)`

Parameters:

| | |
|-----------------------|---|
| <i>SourceImage \$</i> | The (usually smaller) source image in the list. |
| <i>DestImage\$</i> | The larger image, often created with the CreateBitmap command. |
| <i>x1,y1,x2,y2</i> | The insertion coordinates in <i>DestImage\$</i> . |
| <i>Result</i> | 1 if the operation was successful, otherwise 0 if the source or destination image does not exist in the list, or -1 if there is a problem with the coordinates. |

Remarks:

The destination image can be saved with the [SaveBitmap](#) command in the desired format. Coordinate problems include x1 or y1 being negative, or x2,y2 outside the boundaries of the source image, or the *SourceImage\$* coordinates do not match the size of the rectangle defined in *x1,y1,x2,y2*. It is your responsibility to ensure that the coordinates specified are correct.

Related Commands:

[CreateBitmap](#) [DuplicateImage](#) [ExtractListImageRect](#) [SaveBitmap](#)

InstallColorProfile

PIXCL 5 command: Windows 98 or later and Windows 2000 include colour management functions using the standardised profiles in the International Colour Consortium (ICC) format. Colour profiles are stored in the **c:\windows\system\color** directory, and have to be made known to Windows before they can be used. If you add a colour profile, for example, supplied with a colour printer or scanner or other image input device, use this command to install the profile if required.

Syntax: `InstallColorProfile(ProfileName$,Result)`

Parameters:

ProfileName\$ The name of a colour profile, extension **.icm**.
Result 1 if the operation was successful, otherwise 0.

Related Commands:

[UninstallColorProfile](#)

Instr

Returns a number representing the starting position of one string within another string.

Syntax: `Instr(String1$,String2$,Location)`

Parameters:

String1\$ The string to be searched.

String2\$ The string you want to look for in *String1\$*.

Location An integer variable that will contain the starting position of *String2\$* within *String1\$*, or 0 for not found.

Remark:

The first position within *String1\$* is 1.

Example:

This program tests for the presence of "for" within the string "testing for." It then draws the result, 9, at position (10,10) on the screen.

```
Test1$="testing for"  
Test2$="for"  
Instr(Test1$,Test2$,Result)  
DrawNumber(10,10,Result)  
WaitInput()
```

Related Commands:

[Substr](#), [Left](#), [Right](#), [Len](#)

Int

Return the integer part of a floating point number. Updated for **PIXCL 5.1** to support double numbers.

Syntax: `Int(FpNumber&|Fp64#&, Number)`

Parameters:

`FpNumber&` The floating point number. Note that if this number exceeds 32 bit values, the return is 0. Use `Int64` in preference.

`Number` The returned integer.

Related Commands:

[Float](#) [FpVar](#) [FpStr](#) [Int64](#)

Int64

PIXCL 5.1 command. Return the integer part of a floating point or double precision number.

Syntax: `Int64(FpNumber&|Fp64#&, Number64#)`

Parameters:

`FpNumber&|Fp64#&` The floating point number.
`Number64#` The returned 64 bit integer.

Related Commands:

[Float](#) [FpVar](#) [FpStr](#) [Int](#)

InvertRectangle

Inverts the colors of the specified rectangle in the client area. This is not the same as inverting the colors of a bitmap stored in memory, which requires the [InvertImage](#) command.

Syntax: [InvertRectangle\(x1,y1,x2,y2\)](#)

Parameter:

[x1,y1,x2,y2](#) The client area coordinates to be inverted.

Remark:

The region that is inverted will remain inverted until you issue the [InvertRectangle](#) command again, or some other command that writes over the region. The [DrawBackground](#) and [DrawBitmap](#) commands will do this, as will any of the other draw commands. This command can be used for button bitmaps, or to highlight an area of interest in the client area.

Example:

This code fragment draws a sized bitmap at the specified coordinates, then inverts the colors of a sub-region.

```
DrawSizedBitMap(10,10,300,200,Bitmap$)
InvertRectangle(50,60,100,100)
```

Related Command:

[DrawRectangle](#), [DrawEdgeRectangle](#), all the Draw commands, [DrawBitMap](#), [DrawSizedBitmap](#), [DrawBackground](#)

IPAddressBox

When an Internet IP address (e.g. 192.72.14.1) rather than a Web page URL has to be entered, the [IPAddressBox](#) simplifies the process. It is used along with a [Button](#) and/or [SetKeyboard](#) command, plus the [GetIPAddress](#) command.

Syntax: `IPAddressBox(x1,y1,x2,y2,Result)`

Parameters:

x1,y1,x2,y2 The client area coordinates of the control. Setting either of *x2,y2* to 0 removes an existing control.
Result 0 if the control cannot be created (such as when it already exists)

Example:

```
IPAddressBox(100,100,300,130,Res)
Button(310,100,400,130,PUSH,"IP Done",GettingIPAddress)
Goto Wait_for_Input
```

GettingIPAddress:

```
WaitInput(1)
GetIPAddress(P,A,B,C,D)
IPAddressBox(0,0,0,0,Res)
Button()
DrawBackground
DrawNumber(10,40,A) DrawNumber(50,40,B)
DrawNumber(90,40,C) DrawNumber(130,40,D)
```

Related Command:

[GetIPAddress](#)

ItemCount

The number of items in string variable lists with specified delimiter characters can be counted with the [ItemCount](#) command.

Syntax: [ItemCount\(List\\$,Delimiter\\$,Count\)](#)

Parameters:

| | |
|-----------------------------|---|
| List\$ | The delimited list. |
| Delimiter\$ | The defined delimiter character in the list. |
| Count | The number of items in the list. If List is a null string, Count returns 0. |

Related Commands:

[ItemInsert](#), [ItemExtract](#), [ItemLocate](#), [ItemRemove](#)

ItemInsert

A new item string can be inserted into a *List\$* with the *ItemInsert* command.

Syntax: *ItemInsert(List\$,Delimiter\$,Index,Item\$,Result)*

Parameters:

| | |
|--------------------|--|
| <i>List\$</i> | The delimited list string variable . The updated list is returned here. |
| <i>Delimiter\$</i> | The defined delimiter character in the list. |
| <i>Index</i> | The insertion point in the list. |
| <i>Item\$</i> | The new item to insert. |
| <i>Result</i> | 1 if the insert was successful, otherwise 0. |

Remarks:

The defined insertion point is where the new item is placed. Hence, if a new item 3 is inserted, the old item 3 becomes item 4. If, for example, a list has 6 items, and you want to append an item, specify the item number as 7. You can also append or prepend items to any list with a string concatenation function. e.g.

Appending:

```
Delimiter$ = "|"
List$ = List$ + Delimiter$
List$ = List$ + Item$
Goto Wait_for_Input
```

If the insertion index number is greater than the number of items + 1, the function will fail and return 0.

Related Commands:

[ItemCount](#) [ItemInsert](#) [ItemExtract](#) [ItemLocate](#) [ItemRemove](#)

ItemExtract

An item string can be extracted from a [List\\$](#) with the [ItemExtract](#) command.

Syntax: [ItemExtract\(List\\$,Delimiter\\$,Index,Item\\$,Result\)](#)

Parameters:

| | |
|-----------------------------|---|
| List\$ | The delimited list. |
| Delimiter\$ | The defined delimiter character in the list. |
| Index | The extract point in the list. |
| Item\$ | The item extracted. Can be a NULL string. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[ItemCount](#) [ItemInsert](#) [ItemExtract](#) [ItemLocate](#) [ItemRemove](#)

ItemLocate

An item string can be located in a [List\\$](#) with the [ItemLocate](#) command.

Syntax: [ItemLocate\(List\\$,Delimiter\\$,Item\\$,Index\)](#)

Parameters:

| | |
|-----------------------------|--|
| List\$ | The delimited list. |
| Delimiter\$ | The defined delimiter character in the list. |
| Item\$ | The item to be located. |
| Index | The index in the list, otherwise 0. |

Related Commands:

[ItemCount](#) [ItemInsert](#) [ItemExtract](#) [ItemLocate](#) [ItemRemove](#)

ItemRemove

An item string can be removed from *List\$* with the **ItemRemove** command. Use **ItemLocate** to get the *Index* of the list item to remove.

Syntax: `ItemRemove(List$,Delimiter$,Index,Result)`

Parameters:

| | |
|--------------------|--|
| <i>List\$</i> | The delimited list. |
| <i>Delimiter\$</i> | The defined delimiter character in the list. |
| <i>Index</i> | The index of the item to be removed. |
| <i>Result</i> | 1 if the item is removed from <i>List\$</i> , otherwise 0. |

Related Commands:

[ItemCount](#) [ItemInsert](#) [ItemExtract](#) [ItemLocate](#)

JoyGetDevCaps

PIXCL 5 command. Get the number of joysticks attached to the system.

Syntax: `JoyGetDevCaps(JoyStickID,Name$,Buttons,Axes)`

Parameter:

| | |
|-------------------------|---|
| <code>JoyStickID</code> | Either 0 or 1. |
| <code>Name\$</code> | The returned name of the selected joystick. |
| <code>Buttons</code> | The number of available buttons. |
| <code>Axes</code> | The number of axes of movement. |

Related Commands:

[JoyGetNumDevs](#) [JoySetCapture](#) [JoyReleaseCapture](#) [JoyGetThreshold](#)

JoyGetNumDevs

PIXCL 5 command. Get the number of joysticks attached to the system.

Syntax: [JoyGetNumDevs\(*Number*\)](#)

Parameter:

[Number](#) The number of joysticks that have been attached and set up. Maximum is 2.

Related Commands:

[JoyGetDevCaps](#) [JoySetCapture](#) [JoyReleaseCapture](#) [JoyGetThreshold](#)

JoyGetThreshold

PIXCL 5 Command. The movement threshold is the distance the joystick must be moved before your PIXCL application responds. The threshold is initially zero.

Syntax: `JoyGetThreshold(JoyStickID, Threshold)`

Parameter:

JoyStickID Either 0 or 1.
Threshold The current value.

Related Commands:

[JoySetCapture](#) [JoyReleaseCapture](#) [JoyGetNumDevs](#) [JoyGetDevCaps](#) [JoyGetThreshold](#)

JoyReleaseCapture

PIXCL 5 Command. The [joyReleaseCapture](#) command release a joystick.

Syntax: [JoyReleaseCapture\(JoyStickID\)](#)

Parameters:

[JoyStickID](#) Either 0 or 1.

Related Commands:

[JoySetCapture](#) [JoyGetNumDevs](#) [JoyGetThreshold](#) [JoyGetDevCaps](#) [JoyGetThreshold](#)

JoySetCapture

PIXCL 5 Command. The [JoySetCapture](#) command captures a joystick by causing its messages to be sent to the PIXCL Application.

Syntax: [JoySetCapture\(JoyStickID,Period\)](#)

Parameters:

[JoyStickID](#) Either 0 or 1.

[Period](#) The polling period in milliseconds

Related Commands:

[JoyReleaseCapture](#) [JoyGetNumDevs](#) [JoyGetThreshold](#) [JoyGetDevCapsJoySetThreshold](#)

JoySetThreshold

PIXCL 5 Command. The movement threshold is the distance the joystick must be moved before your PIXCL application responds. The threshold is initially zero.

Syntax: `JoySetThreshold(JoyStickID,Threshold)`

Parameter:

JoyStickID Either 0 or 1.
Threshold The new desired value.

Related Commands:

[JoySetCapture](#) [JoyReleaseCapture](#) [JoyGetNumDevs](#) [JoyGetDevCaps](#) [JoyGetThreshold](#)

LCase

Converts a string to lowercase.

Syntax: `LCase(String$)`

Parameter:

`String$` The string to convert.

Example:

This program reads the text on the Clipboard, converts it to lowercase, and then writes it back out to the Clipboard.

```
ClipboardGet (String$, Result)  
LCase (String$)  
ClipboardPut (String$, Result)
```

Related Command:

[UCase](#)

Left

Returns a specified number of characters from the left of a string.

Syntax: `Left(String$,Places,Result$)`

Parameters:

String\$ The string you want to extract the text from.

Places The number of places you want.

Result\$ A string variable that will contain the result.

Example:

This program extracts the first word from the string " Ottawa Canada " and draws it at point (10,10) in the PIXCL window.

```
String$="Ottawa Canada"  
Instr(String$," ",Location)  
Location = Location - 1  
Left(String$,Location,Result$)  
DrawText(10,10,Result$)  
WaitInput()
```

Related Commands:

[Right](#), [RightOf](#), [Instr](#), [Substr](#)

LeftOf

Returns characters to the left of a location in a string.

Syntax: `LeftOf(String$,Location,Result$)`

Parameters:

| | |
|-----------------|---|
| <i>String\$</i> | The string you want to extract the text from. |
| <i>Location</i> | The location in the string. The character at the specified location is not returned. |
| <i>Result\$</i> | A string variable that will contain the result. It can be the same as the input string. |

Example:

This program returns the first word from the string " Ottawa Canada " and draws it at point (10,10) in the PiXCL window.

```
String$="Ottawa Canada"  
Instr(String$," ",Location)  
LeftOf(String$,Location,Result$)  
DrawText(10,10,Result$)  
WaitInput()
```

Related Commands:

[Right](#) [RightOf](#) [Instr](#) [Substr](#)

Len

Returns the length of a string in bytes.

Syntax: `Len(String$,Length)`

Parameters:

String\$ The string whose length you want to determine.

Length An integer variable that will contain the length.

Example:

This example puts up a text box asking you to enter your Zipcode. If the text you enter is longer than 10 bytes, a message box appears indicating the entry is too long. The program then loops back for you to try again.

```
Text$="Please enter your Zipcode"
Caption$="Enter ZIP"
Box:
  TextBox(Text$,Caption$,Input$,ButtonPushed)
  If ButtonPushed = 0 Then End
  Len(Input$,Length)
  If Length <= 10 Then Goto Wait_for_input
  BoxText$ = "Zipcode too long"
  MessageBox(OK,1,INFORMATION,BoxText$,"Error",Button)
Goto Box:
  Wait_for_input:
WaitInput()
```

Related Commands

[Right](#) [Instr](#) [Substr](#) [Left](#)

ListBox

Puts up a dialog box with a list box inside. The list box gets its contents from a string variable you provide. Only one list item can be selected with this dialog. If the list has more items than can be displayed, a vertical scrollbar is created automatically. If one of the list items is longer than thirty characters, a horizontal scrollbar is created automatically. The PiXCL 5 USListBox command is identical, except that the list is presented unsorted.

Syntax: `ListBox(Caption$, List$, Delimiter$, Result$)`

PIXCL 5: `USListBox(Caption$, List$, Delimiter$, Result$)`

Parameters:

| | |
|--------------------|---|
| <i>Caption\$</i> | The text you want to appear in the title bar of the dialog box. |
| <i>List\$</i> | A string containing the items you want to appear in the list box. |
| <i>Delimiter\$</i> | The character you've used to delimit the items in <i>List\$</i> . |
| <i>Result\$</i> | A string variable that will contain the item you select. If you choose Cancel (or press ESC) to leave the dialog box, <i>Result\$</i> is assigned a null string (""). |

Remarks:

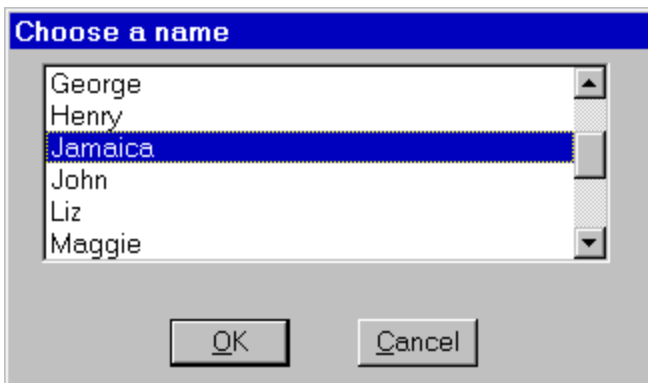
The items in *List\$* must be separated from one another (delimited) using the character in *Delimiter\$*. You can use any character you want as the delimiter; a space or semicolon will usually suffice.

You can select an item from the list box by highlighting it and choosing OK or double-clicking on it.

Windows always sorts the items alphabetically before displaying them in the list box.

Example:

This example creates a list of space-delimited names and displays them in a list box, as shown in the figure below. After you select a name and choose OK, your selection appears in a message box.



A ListBox dialog box.

```
Caption$ = "Choose a name"  
List$ = "Jamaica Sammy Millie Maggie George Elise John Mary "  
List$ = List$ + "Dorothy Noel Carrie Burt Liz Tom Henry Richard"  
ListBox(Caption$,List$," ",Result$)  
If Result$ = "" Then Out$ = "You chose Cancel" | Goto Message
```

```
Out$ = "You picked " + Result$  
Message:  
MessageBox(OK,1, INFORMATION,Out$,"ListBox results",Button)  
WaitInput()
```

Related Commands:

[DialogBox](#) [FileGet](#) [TextBox](#) [EnumWindows](#)

ListBoxExt

Puts up a dialog box with a multi-column list box inside. This extended list box gets its contents from a string variable you provide, and includes a third button labeled "Help". Multiple list items can be selected with this dialog. Vertical and horizontal scrollbars are created automatically as needed.

Syntax: `ListBoxExt(Label$,List$,Delim$,Help$,Res$)`

Parameters:

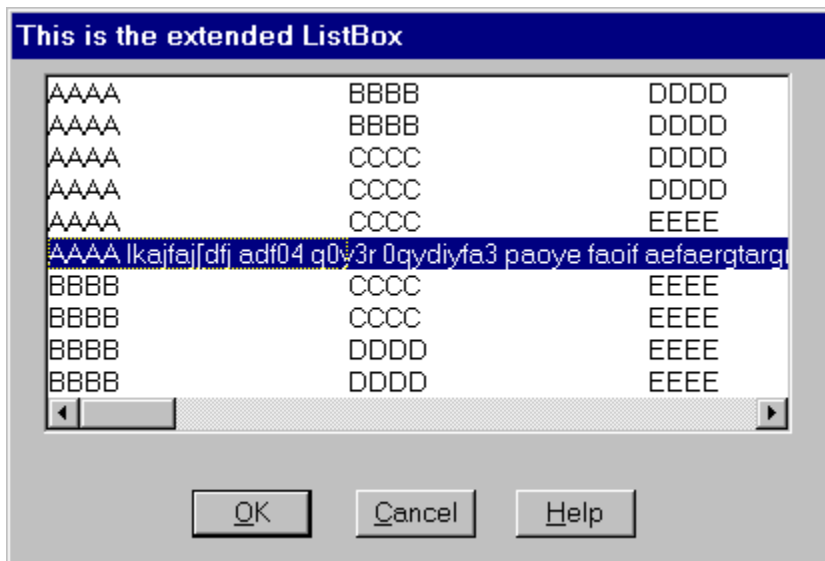
| | |
|-----------------|---|
| <i>Label\$</i> | The text you want to appear in the title bar of the dialog box. |
| <i>List\$</i> | A string containing the items you want to appear in the list box. |
| <i>Delim\$</i> | The character you've used to delimit the items in <i>List\$</i> . |
| <i>Help\$</i> | The text that appears in a messagebox if the Help button is pressed. |
| <i>Result\$</i> | A string variable that will contain the item you select. If you choose Cancel (or press ESC) to leave the dialog box, <i>Result\$</i> is assigned a null string (" "). Multiple selections are delimited with the <i>Delim\$</i> character above. |

Remarks:

The items in *List\$* must be separated from one another (delimited) using the character in *Delimiter\$*. You can use any character you want as the delimiter; a space or semicolon will usually suffice.

You can select an item or items from the list box by highlighting it or them, and choosing OK or double-clicking on it. Use the Shift and Control modifier keys as well.

Windows always sorts the items alphabetically before displaying them in the list box.



A ListBoxExt dialog box.

Related Commands:

[DialogBox](#) [FileGet](#) [TextBox](#) [EnumWindows](#)

ListLoadedBitmaps

PiXCL maintains an internal linked list of any images loaded with the [DrawBitmap](#) command set. This command can be used to check on the list and take appropriate action. With the [DrawPreviewBitmap](#) and [LoadBitmap](#) commands, you can also load a preview image. This will also appear in the list as two entries of the same name. In the list itself, each image is identified as a Full or Preview image. Using [FreeBitmap](#) will remove both types if they exist.

Syntax: `ListLoadedBitmaps(List$,Delimiter$,Count)`

Parameters:

- List\$** The list of bitmap filenames is returned in *List\$*. Null if no bitmaps are loaded.
- Delimiter\$** The character that you want use a list item delimiter. “|” is often suitable, or you could use `Chr(13,CR$)` to create a carriage return delimiter, as per the example below.
- Count** The number of valid entries in the bitmap list. If no images have been loaded, or the [FreeBitmapAll](#) command has been used, *Count* returns 0.

Example:

This code fragment accesses the list and returns a variable that is then displayed in a [MessageBox](#). You could also pass the result to a [ListBox](#) command for display and item selection.

```
CountTheList:
    Chr(13,CR$)
    ListLoadedBitmaps(List$,CR$,Count)
    Str(Count,Count$)
    Msg$ = "Entries in PiXCL Bitmap List = " + Count$
    MessageBox(OK,1,INFORMATION,List$,Msg$,Res)
    Goto Wait_for_Input
```

Related Commands:

[DrawBitmap](#) [DrawSizedBitmap](#) [DrawZoomedBitmap](#) [FreeBitmap](#) [FreeBitmapAll](#) [LoadBitmap](#)

LoadBitmap

This command loads an image into memory without displaying it. It is equivalent to a [DrawSizedBitmap\(0,0,0,0,ImageFile\\$\)](#) command. It is up to you to ensure that the image exists on disk before you try to load it.

Syntax: [LoadBitmap\(ImageFile\\$,PREVIEW | FULL\)](#)

Parameters:

| | |
|-----------------------------|--|
| ImageFile\$ | The name of the image file to load. |
| FULL | Load the whole image. |
| PREVIEW | Load a preview image that has maximum dimensions |

Related Commands:

[DrawBitmap](#) [DrawSizedBitmap](#) [FreeBitmap](#) [FreeBitmapAll](#)

LoadBitmapExt

This command checks the image format, and if recognized, loads and image into memory without displaying it. It is equivalent to a [DrawSizedBitmap\(0,0,0,0,ImageFile\\$\)](#) command. It is up to you to ensure that the image exists on disk before you try to load it.

Syntax: `LoadBitmapExt(ImageFile$,PREVIEW | FULL,Result)`

Parameters:

| | |
|-----------------------------|---|
| ImageFile\$ | The name of the image file to load. |
| FULL | Load the whole image. |
| PREVIEW | Load a preview image that has maximum dimensions |
| Result | 1 if the imagefile can be loaded, otherwise 0 if the image format is unknown, or the image cannot be found. |

Related Commands:

[DrawBitmap](#) [DrawSizedBitmap](#) [FreeBitmap](#) [FreeBitmapAll](#) [CheckBitmapFormat](#)

LoadStdProfileSettings

PIXCL 5 command. All new applications should keep program parameters in the Registry. The [LoadStdProfileSettings](#) command (for now) just stores the last window position in screen coordinates.

Syntax: [LoadStdProfileSettings\(DeveloperName\\$,AppName\\$,sx1,sy1,sx2,sy2,Result\)](#)

Parameters:

| | |
|---------------------------------|---|
| DeveloperName\$ | The name of the application developer or company name. This reads from the HKEY_CURRENT_USER tree, Software\DeveloperName\$\AppName\$\Settings\WindowPos key. |
| AppName\$ | The name of your application. |
| sx1,sy1,sx2,sy2 | The screen coordinates of the application when it was last run. These are decoded from the string read from the Registry. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Command:

[SaveStdProfileSettings](#)

LoadImageColorMap

PIXCL 5 command. For 8 bit images loaded into the image list, you can replace the current colour map with a new palette from a (**.PAL**) file.

Syntax: `LoadImageColorMap(ListImageName$, FULL|PREVIEW,PALfile$,Result)`

Parameters:

| | |
|------------------------|--|
| <i>ListImageName\$</i> | The name of the image loaded into your PiXCL application. This must be 8 bits per pixel, or Result returns 0. 24 bit images do not have a colour map. |
| <i>FULL PREVIEW</i> | Defines whether the image was loaded in full or preview mode. In most instances, you will load the image in FULL mode eg with DrawBitmap . |
| <i>PALfile\$</i> | The name of the file to be read. If the file does not exist, the call fails. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[SaveImageColorMap](#) [CreatePALfile](#) [DrawBitmap](#)

Log10

Floating Point math library function. Calculate the base 10 logarithmic function **Log(x)**.

Syntax: `Log10(X&, Value&)`

Parameters:

`X&` The X value to use.

`Value&` The result of the function.

Related Commands:

[Exp](#) [_LogE](#)

LogE

Floating Point math library function. Calculate the Naperian log function **LogE(x)**.

Syntax: `LogE(X&, Value&)`

Parameters:

`X&` The X value to use.

`Value&` The result of the function.

Related Commands:

[Exp](#) [Log10](#)

Logoff

Logs out of Windows by closing all running programs including Explorer. The task bar will disappear, and the Login dialog will eventually be displayed by Windows.

Syntax: [Logoff](#)

Remark:

Logoff is the equivalent of selecting **Start: Log Off** <current_use_name>

Before logging off, Windows polls all active applications to see if it's OK to close them. If you haven't yet saved your work, an application can interrupt the shutdown and ask you to save.

Example:

This example asks whether you want to log off Windows. If you select Yes, it executes the Logoff command.

```
MessageBox(YESNO,2,QUESTION,"Log off Windows?",  
           "Logoff?",ButtonPushed)  
If ButtonPushed = 1 Then Logoff
```

Related Commands:

[ExitWindows](#) [Shutdown](#) [AbortShutdown](#)

Max

Floating Point math library function. Locate the maximum of a list of real numbers.

Syntax: $\text{Max}(\text{Number}_1\&, \dots, \text{Number}_n\&, \text{Value}\&)$

Parameters:

$\text{Number}\&$ Each of the list of numbers. These numbers can be integers (not integer variables).
 $\text{Value}\&$ The maximum of the list.

Related Commands:

[Average](#), [Min](#)

MCIGetErrorString

PIXCL 5 command. The [MCISendString](#) command will return an error code if the operation requested fails for some reason. This code can be converted into an error description with the [MCIGetErrorString](#) command.

Syntax: [MCIGetErrorString](#)(*ErrorCode*, *ErrorString\$*)

Parameters:

[ErrorCode](#) The code returned by [MCISendString](#).
[ErrorString\\$](#) The error description.

Related Commands:

[MCISendString](#)

MCISendString

PIXCL 5 command. Multimedia devices are usually supported by the Multimedia Command Interface or MCI. This command implements the MCI command string interface, where the command is assembled into a string and passed to the device. The result is also returned in a string. Your PIXCL application has to parse and convert the results into integers or smaller strings as might be required.

Syntax: `MCISendString(Command$, Result$, ErrorCode)`

Parameters:

| | |
|------------------|---|
| <i>Command\$</i> | The command string. Exact syntax is dependent on the device. Check your device documentation and the MCI Command Strings Reference. |
| <i>Result\$</i> | The result of the command, as a string. This might include integer values, or corners of a rectangle, or other information specific to the MCI driver. Check your device documentation. |
| <i>ErrorCode</i> | 1 if the operation was successful, otherwise a value that can be passed to the MCIGetErrorString command to be decoded. |

Related Commands:

[MCIGetErrorString](#) [MCI Command Strings](#)

Min

Floating Point math library function. Locate the minimum of a list of real numbers.

Syntax: `Min(Number_1&, ..., Number_n&, Value&)`

Parameters:

`Number&` Each of the list of numbers. These numbers can be integers (not integer variables).
`Value&` The minimum of the list.

Related Commands:

[Average](#), [Max](#)

MessageBeep

Plays the waveform sound file associated with an entry in the [sounds] section of WIN.INI.

Syntax:

`MessageBeep(BEEP/ASTERISK/EXCLAMATION/HAND/QUESTION/OK)`

Parameters:

| | |
|-----------------------------|---|
| BEEP | Issues the standard beep using the computer speaker. |
| ASTERISK | Plays the file associated with the SystemAsterisk setting. |
| EXCLAMATION | Plays the file associated with the SystemExclamation setting. |
| HAND | Plays the file associated with the SystemHand setting. |
| QUESTION | Plays the file associated with the SystemQuestion setting. |
| OK | Plays the file associated with the SystemDefault setting. |

Remarks:

Here are some typical entries in the [sounds] section of WIN.INI:

```
[sounds]
SystemDefault=ding.wav, Default Beep
SystemExclamation=ding.wav, Exclamation
SystemStart=chimes.wav, Windows Start
SystemExit=chimes.wav, Windows Exit
SystemHand=ding.wav, Critical Stop
SystemQuestion=ding.wav, Question
SystemAsterisk=ding.wav, Asterisk
```

If you use the command `MessageBeep(ASTERISK)`, for example, PiXCL plays the sound file identified by the SystemAsterisk entry, which in this case is ding.wav.

Before PiXCL can play a sound file, you must first install a supported sound board and its associated waveform audio device driver.

The `MessageBeep` command has its origin in Windows 3.x. For a more fully featured sound function that lets you play waveform sound files or sound entries in the Windows registry, see the [WAVPlaySound](#) command.

Example:

This example displays a message telling you to press ESC if you want to end the current program and plays the sound associated with the SystemExclamation setting in WIN.INI. It then waits for a keystroke.

```
{Draw message on screen and play EXCLAMATION sound}
  DrawText(1,1,"Press ESC to end the program")
  MessageBeep(EXCLAMATION)

{Set up keyboard}
  SetKeyboard(27,Esc)      {27=virtual key for ESC}
```

```
Wait_for_input:  
    WaitInput()
```

```
Esc:  
    End
```

Related Commands:

[WAVGetDevCaps](#), [WAVGetNumDevs](#), [WAVGetPitch](#), [WAVPlaySound](#), [WAVSetPitch](#), [WAVSetPlayRate](#)

MessageBox

This is one of the most used PiXCL commands, and creates a custom message box with your own prompt, caption, and one, two or three push buttons. MessageBoxes can display one of the system-defined icons (Application, Information, Exclamation, Question, Stop and Windows Logo) that you are familiar with in other Windows software, or any of the twenty-one icons built into PiXCL as well.

Messageboxes appear centralized in the screen area. If you want to develop more advanced messageboxes positioned relative to the PiXCL client area, see the [DialogBox](#) command.

Syntax:

`MessageBox(TYPE,DefaultButton,ICON,Text$,Caption$, ButtonPushed)`

Parameters:

| | |
|----------------------------|--|
| <code>TYPE</code> | Specifies the type of push buttons that appear within the message box. It must be one of the tokens in the table below. |
| <code>DefaultButton</code> | An integer indicating the button you want to appear as the default. The buttons are numbered from left to right starting with 1. |
| <code>ICON</code> | Controls the type of icon that appears in front of <code>Text\$</code> in the message box. You must use one of the tokens in the table below. |
| <code>Text\$</code> | The message to be displayed within the message box. <code>Text\$</code> can occupy multiple lines in the script (see the second example below). You can have as many lines as you like, but be aware that if you have too many, the message box may not fit on the screen. |
| <code>Caption\$</code> | The text you want to place at the top of the message box. |
| <code>ButtonPushed</code> | An integer variable that returns a number corresponding to the button that was pushed to leave the dialog box. The buttons are numbered from left to right starting with 1. |

Push Button Types

| <u>Token</u> | <u>Meaning</u> |
|-------------------------------|---|
| <code>OK</code> | Displays one push button: OK. |
| <code>OKCANCEL</code> | Displays two push buttons: OK and Cancel. |
| <code>RETRYCANCEL</code> | Displays two pushbuttons: Retry and Cancel |
| <code>YESNO</code> | Displays two push buttons: Yes and No. |
| <code>YESNOCANCEL</code> | Displays three push buttons: Yes, No, and Cancel. |
| <code>ABORTRETRYIGNORE</code> | Displays three push buttons: Abort, Retry and Ignore. |

MessageBox Icon Types

| <u>Token</u> | <u>Meaning</u> |
|--------------------------|--|
| <code>INFORMATION</code> | Displays an icon consisting of a lowercase <code>i</code> in a circle. |

| | |
|-------------|---|
| EXCLAMATION | Displays an exclamation-point icon. |
| QUESTION | Displays a question-mark icon. |
| STOP | Displays a stop sign icon. |
| APP | Displays a generic application logo that looks like a small dialog box. |
| WINLOGO | Displays the Windows logo. |
| NOICON | Displays no icon in the MessageBox |



The above icon styles, in order.

ICON01 - ICON19 Displays one of the icons built into PiXCL. These are the same tokens that are used in the [DrawIcon](#) and [DrawIconFile](#) commands. Tokens [PXLHISTOGRAM](#), [PXLTOOLBAR](#), [SCANNER](#), [DIGICAM](#) and [SCANCAM](#) also display usable icons in the MessageBox.

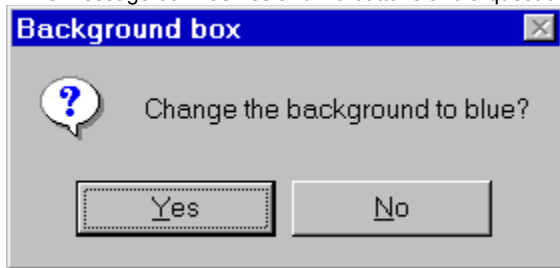
Remarks:

Most of the time you will use the system icons for your messageboxes. It is preferable to keep messagebox text short, or else your users will generally ignore them. For error messages, try to ensure that the severity of the message is apparent, and that you provide information on how to correct the problem.

If you need to provide more details, it is better done with a call to [WinHelp](#) to display a suitable help topic.

Examples:

The following program displays a message box that asks whether you want to change the background of the window to blue. The message box has Yes and No buttons and a question-mark icon, as shown below.



A sample message box.

```

MessageBox(YESNO,1,QUESTION,
"Change the background to blue?",
"Background box",Button)
If Button = 1
    UseBackground(OPAQUE,0,0,255)
    DrawBackground
Endif
WaitInput()

```

This next example displays three message boxes, each with more lines of text than the previous one.

```
{Show a single-line message}
    MessageBox(OKCANCEL,1,QUESTION,
    "See a message box with a two-line message?",
    "One-line message",Button_mashed)
    If Button_mashed=2 Then Goto Wait_for_input

{Show a two-line message}
    MessageBox(YESNO,1,QUESTION,
    "This box has two lines of text.
    Do you want to see one with three?",
    "Two-line message",Button_mashed)
    If Button_mashed=2 Then Goto Wait_for_input

{Show a three-line message}
    MessageBox(OK,1,INFORMATION,

    "This box has three lines of text.
    You've now seen message boxes with,
    one, two, and three lines of text.",
    "Three-line message",Button_mashed)

Wait_for_input:
    WaitInput()
```

Remarks:

The text string in any message box can be any length you want, if you include carriage returns in the script. In practice, a message box should have enough information to allow the user to make a suitable choice. Information or Exclamation messageboxes can also be used as simple Help dialogs.

Related Commands:

[DrawIcon](#) [MessageBeep](#) [WinHelp](#) [WinHTMLHelp](#)

MonitorFromPoint

For multiple monitor systems, you can find which monitor is the nearest to a point.

Syntax: `MonitorFromPoint(X, Y, MonitorName$)`

Parameters:

`X, Y` A point in the virtual desktop (not an application client area).
`MonitorName$` The name of the nearest monitor to or containing the point.

Related Commands:

[EnumDisplayMonitors](#) [MonitorFromRect](#) [MonitorFromWindow](#)

MonitorFromRect

For multiple monitor systems, you can find which monitor is the nearest to a rectangle. This would typically be the coordinates of a window.

Syntax: `MonitorFromRect(X1, Y1, X2, Y2, MonitorName$)`

Parameters:

`X1, Y1, X2, Y2` A rectangle in the virtual desktop (not an application client area).
`MonitorName$` The name of the nearest monitor to or containing the rectangle.

Related Commands:

[EnumDisplayMonitors](#) [MonitorFromPoint](#) [MonitorFromWindow](#)

MonitorFromWindow

For multiple monitor systems, you can find which monitor contains an application main window.

Syntax: `MonitorFromWindow(WindowName$, MonitorName$)`

Parameters:

`WindowName$` A main window title.

`MonitorName$` The name of the nearest monitor to or containing the application window.

Related Commands:

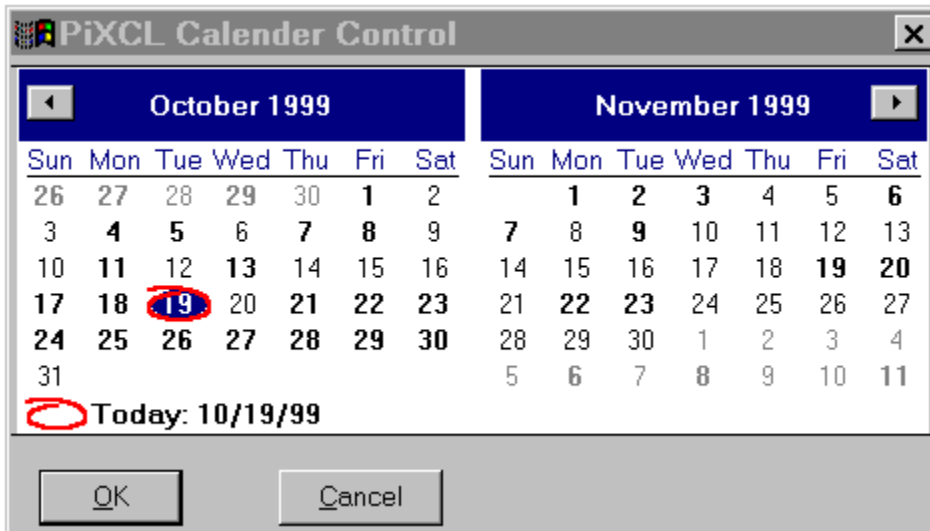
[EnumDisplayMonitors](#) [MonitorFromPoint](#) [MonitorFromRect](#)

MonthCalControl

The MonthCalControl is a control that is visible and active until you explicitly close it, while allowing all other main PiXCL application window functions to be available. It provides a simple and intuitive way for you to select a date from a familiar interface.

The control window is owned by your PiXCL application, so that if the application exits, it automatically closes the control window as well.

The MonthCalControl command requires **shell32.dll** version 4.70 or later. If you have Windows 95 or NT 4 with Internet Explorer 4.01 or later installed, or are running Windows 98 or later, this command should work on your system. If an earlier version of shell32.dll (e.g. 4.00) is installed, this command has no effect.



The left and right hand arrows change the month up or down.

Syntax: `MonthCalControl(X,Y, Title$, MonthDelta, TOKEN_1, TOKEN_2, TOKEN_3, TOKEN_4, Date_TOKEN, TimeString$, Result)`

Parameters:

`X,Y` The top left corner client area coordinates for the control.

`Title$` The string that appears in the title bar of the control.

`MonthDelta` The number of months to display, and the month delta value. The default is one. The control automatically adjusts its size according to `MonthDelta`.

`TOKEN_1_2_3` in order
`(NO)TODAY` displays a "today" string at the bottom of the control.
`(NO)TODAYCIRCLE` displays a circle around today's date.
`(NO)WEEKNUMBERS` displays week numbers adjacent to the month display.

`TOKEN_4` `(NO)MOVE` defines whether the control can be moved within the PiXCL application client area. If `NOMOVE` is specified, the titlebar is removed, and the value in `Title$` is ignored. You can set `Title$` to an empty string if you wish.

`date_TOKEN` All example dates below are the same. These are the same tokens as in the `TimeToASCII` command, and in fact, the same internal code is executed to do the conversion.

MMDDYYYY e.g. 2/8/1999
DDMMYYYY e.g. 8/2/1999
WDDDMYYYY e.g. Saturday, 8 February, 1999
WDMDDYYYY e.g. Saturday, February 8, 1999
MDDYYYY e.g. February 8, 1999

TimeString\$ The returned string of the current selected day, in the above `date_TOKEN` format. If the **Cancel** button is pressed, **TimeString\$** is unchanged from its current value, which may be an empty string.

Result 1 if the OK button was pressed, otherwise 0.

The Month Calendar Control User Interface

The month calendar control interface allows you to select a date from the displayed days or change the control's display in various ways.

- Scrolling the control's display.

By default, when you click the arrow buttons in the top left or top right of the month calendar control, it updates its display to show the previous or next month. If the month calendar control is displaying more than one month at a time, the display changes by the number of months currently in view. That is, if the month calendar displays January, February, and March and the user clicks the top right arrow button, the control updates its display to show April, May, and June. You can also perform the same action by clicking the partial months displayed before the first month and after the last month.

- Selecting a nonadjacent month.

When you click the name of a displayed month, a pop-up menu appears that lists all months within the year. The user can select a month on the list. If the user's selection is not visible, the month calendar control scrolls its display to show the chosen month.

- Selecting a different year.

If you click the year displayed next to a month name, an up-down control appears in place of the year. You can change the year with this control. The month calendar control updates its display for the selected year when the up-down control loses focus.

-Selecting the current day.

If a month calendar control is not using the NOTODAY style, you can return to the current day by clicking the "today" text at the bottom of the control, or right clicking to display a "Go to Today" popup. If the current day is not visible, the control updates its display to show it.

Related Commands:

[TimeToASCII](#)

Negate

Returns the arithmetic negation of the input number. Hence a positive number returns a negative, and a negative number returns a positive.

Syntax: `Negate(Number)`

Parameters:

Number The input and negated output integer number.

Example:

`Negate(Number)`

NumToHex

This command converts a 32 bit integer to the equivalent hexadecimal string.

Syntax: `NumToHex(Number,HexString$)`

Parameters:

Number Any positive or negative 32 bit integer.

HexString\$ The returned hexadecimal string.

Example:

Convert:

```
NumToHex(6523881,Hex$)
DrawText(10,10,Hex$)
Number = 6523881
NumToHex(Number,Hex$)
DrawText(10,30,Hex$)
Goto Wait_for_Input
```

Related Commands:

[HexToNum](#), [Str_Val](#)

PackRGB

PIXCL 5 command. It can often be handy to pack RGB color values into an integer and store it in an integer or integer array variable.

Syntax: *PackRGB(Red, Green, Blue, PackedColour)*

Parameters:

Red, Green, Blue Integer values for the defined colour.

PackedColour The packed colour value, in hex format, 0x00bbgrr

Related Commands:

[UnpackRGB](#) [UnpackRGBA](#)

PackRGBA

PIXCL 5 command. It can often be handy to pack RGBA color values into an integer and store it in an integer or integer array variable.

Syntax: *PackRGBA(Red, Green, Blue, Alpha, PackedColour)*

Parameters:

Red, Green, Blue, Alpha Integer values for the defined colour.

PackedColour The packed colour value, in hex format, 0xaabbggr

Related Commands:

[UnpackRGB](#) [UnpackRGBA](#)

Pad

Adds a specified number of spaces to the end of a string.

Syntax: `Pad(String$,Length)`

Parameters:

`String$` A string variable containing the string you want to pad.

`Length` The number of spaces you want to add to the end of `String$`.

Remarks:

Adding spaces to the end of a string can be handy when you want the string to overwrite another longer string that is already displayed in the PiXCL window (see the example).

If you want to add spaces to the start of a string, use the + operator to concatenate two strings together, as in `Variable$ = "+"Cranberry`.

Example:

This program illustrates the benefit of padding a string with spaces before drawing it on the screen. First, a seven-character string is placed on the screen. Next, a shorter string is drawn at the same location. Because a portion of the longer string still remains in view, the shorter string is then padded with spaces and redrawn. The beneficial effect of the padding is that the remainder of the longer string is overwritten.

```
{Draw 7-character string on the screen and pause}
  Text$ = "Testing"
  DrawText(10,10,Text$)
  WaitInput(2000)
{Draw a shorter string at the same location}
  Text$ = "123"
  DrawText(10,10,Text$)
  WaitInput(2000)
{Pad shorter string with spaces and redraw}
  Pad(Text$,10)
  DrawText(10,10,Text$)
  WaitInput()
```

Related Commands:

[Space](#), [Trim](#), [Set](#)

PasswordBox

Visually this is quite similar to the dialog produced by the `TextBox` command, but includes the current default icon. It would be used when you want to input a secure string such as a password. Each character typed is replaced with a '*'.

Syntax: `PasswordBox(Title$,Text$,Btn1$,Btn2$,Btn,Password$)`

Parameters

| | |
|-------------------|---|
| <i>Title\$</i> | The string that appears in the titlebar. |
| <i>Text\$</i> | Up to four lines of text. |
| <i>Btn1\$</i> | Label that appears on the left button. If set to "", the default label is "&Accept". |
| <i>Btn2\$</i> | Label that appears on the right button. If set to "", the default label is "&Reject". |
| <i>Btn</i> | Returned code 1 (left) or 2 (right). |
| <i>Password\$</i> | The returned string. If button 2 was pressed this ALWAYS returns a NULL string, even if the string had been previously defined. |

Related Commands:

[DialogBox](#) [MessageBox](#) [TextBox](#) [TextBoxExt](#)

PathAddBackslash

Adds a backslash to the end of a string to create the correct syntax for a path. If the source path already has a trailing backslash, no backslash will be added.

Syntax: `PathAddBackslash(Path$)`

Parameter:

Path\$ A path string variable.

Related Commands:

All the Path commands, string commands.

PathAddExtension

Adds a file extension to a filename string. If there is already a file extension present, no extension will be added. If the path is an empty string, the result will be the file extension only.

Syntax: `PathAddExtension(Extension$, FilePath%)`

Parameter:

Extension\$ An extension string.e.g. “.txt”
FilePath\$ A path string variable.

Related Commands:

All the Path commands, string commands.

PathAppend

A path can be extended. For example, if the original path string is **name_1\name_2**, and the part to append to end is **name_3**, the appended path string is **name_1\name_2\name_3**.

Syntax: `PathAppend(More $, Path $)`

Parameter:

More \$ A string that gets added to *Path* \$.

Path \$ A path string variable.

Related Commands:

All the Path commands, string commands.

PathCanonicalize

This function allows you to specify what to remove from a path by inserting special character sequences into the path. The ".." sequence indicates to remove the path part from the current position to the previous path part. The "." sequence indicates to skip over the next path part to the following path part. The root part of the path cannot be removed.

Syntax: `PathCanonicalize(SourcePath$, DestPath$)`

Parameter:

`SourcePath$` A string that gets added to `Path$`.
`DestPath $` A path string variable.

Example:

```
SomePath$ = "C:\PiXCLTools\Learning\More\New\program.pxl"  
SourcePath$ = "C:\PiXCLTools\..\More\.\program.pxl"  
PathCanonicalize(SourcePath$, DestPath$)  
DebugMsgBox (DestPath$)  
  
DestPath$ becomes "C:\PiXCLTools\More\program.pxl"
```

Related Commands:

All the Path commands, string commands.

PathCombine

This command concatenates two strings that represent properly formed paths into one path, as well as any relative path pieces.

The directory path should be in the form of **A:B:**, ..., **Z:**. The file path should be in a correct form that represents the file part of the path. The file path must not be a null string, and if it ends with a backslash, the backslash will be maintained.

Syntax: `PathCombine(Directory$, File$, NewPath$)`

Parameter:

| | |
|---------------------------|--|
| <code>Directory \$</code> | The source directory. |
| <code>File\$</code> | A filename. |
| <code>NewPath \$</code> | A path string variable that holds the new file path. |

Related Commands:

All the Path commands, string commands.

PathCommonPrefix

Compares two paths to determine if they share a common prefix. A prefix is one of these types: "C:\", ".", "..", "..\".

Syntax: `PathCommonPrefix(First$, Second$, Common$)`

Parameters:

| | |
|-----------------|---|
| <i>First\$</i> | The first path string. |
| <i>Second\$</i> | The second path string. |
| <i>Common\$</i> | The common prefix, if any, otherwise a null string. |

Related Commands:

All the Path commands, string commands.

PathCompactPath

Truncates a file path to fit within a given pixel width by replacing path components with ellipses. This command is often used to fit a string within an edit control in a dialog box.

Syntax: `PathCompactPath(Path$,Width,CompactPath$)`

Parameters:

Path\$ The input path string.
Width The width in pixels of the region where the path string is to be drawn.
CompactPath\$ The resulting compact path. This works on the basis of the currently selected font.

Related Commands:

All the Path commands, string commands.

PathFindNextComponent

Parses a path for the next path component. Paths are delimited by backslashes or by the end of the path.

Syntax: `PathFindNextComponent (Path$, NextComponent$)`

Parameters:

Path\$ The input path string.

NextComponent\$ The next component of the path..

Related Commands:

All the Path commands, string commands.

PathFindOnPath

Searches for a file on the specified path. This command is similar to [FileExist](#).

Syntax: `PathFindOnPath (File$, Path$, Result)`

Parameters:

| | |
|---------------|--|
| <i>File\$</i> | The input filename to be located. |
| <i>Path\$</i> | The input path string. |
| <i>Result</i> | 1 if File\$ is found on the path, otherwise 0. |

Related Commands:

All the Path commands, string commands, [FileExist](#)

PathsDirectory

Checks if a file is a directory.

Syntax: `PathsDirectory(Path$, Result)`

Parameters:

Path\$ The input path string.

Result 1 if the file is a directory, otherwise 0.

Related Commands:

All the Path commands, string commands.

PathIsSystemFolder

Checks if a file is a system folder.

Syntax: `PathIsSystemFolder(Path$, Result)`

Parameters:

Path\$ The input path string.

Result 1 if the file is a system folder, otherwise 0.

Related Commands:

All the Path commands, string commands.

PathMakeSystemFolder

Set a directory as a system folder.

Syntax: [PathMakeSystemFolder\(Path\\$, Result\)](#)

Parameters:

[Path\\$](#) The input path string.

[Result](#) 1 if the attributes are set, otherwise 0.

Related Commands:

[PathUnmakeSystemFolder](#) All the Path commands, string commands.

PathQuoteSpaces

For paths that include spaces, it is necessary to enclose the path with quotation marks.

Syntax: `PathQuoteSpaces(Path$)`

Parameters:

`Path$` The path to be quoted, and the result.

Related Commands:

All the Path commands, string commands.

PathUnquoteSpaces

Removes any quotation marks around a path string.

Syntax: [PathUnquoteSpaces\(Path\\$\)](#)

Parameters:

[Path\\$](#) The path to be quoted, and the result.

Related Commands:

All the Path commands, string commands.

PathUnmakeSystemFolder

Set a system folder as a normal directory.

Syntax: [PathUnmakeSystemFolder\(Path\\$, Result\)](#)

Parameters:

[Path\\$](#) The input path string.

[Result](#) 1 if the attributes are reset, otherwise 0.

Related Commands:

[PathMakeSystemFolder](#) All the Path commands, string commands.

PixelsToDlgUnits

Converts a pixel coordinate to a dialog units coordinate.

Syntax: [PixelsToDlgUnits\(Px,Py,Dx,Dy\)](#)

Parameters:

[Px,Py](#) The returned pixel coordinate.

[Dx,Dy](#) The dialog box coordinate.

Remarks:

Converting between pixels and dialog coordinates will result in some loss of precision.

Related Commands:

[DialogBox](#) [DlgUnitsToPixels](#) [GetDialogUnits](#)

Pow

Floating Point math library function. Calculate the power function $y^{**}x$.

Syntax: `Pow(Y&, X&, Value&)`

Parameters:

| | |
|-------------------------|-----------------------------|
| <code>Y&</code> | The Y value |
| <code>X&</code> | The X value |
| <code>Value&</code> | The result of the function. |

Related Commands:

[Log10](#) [LogE](#)

PrintBitmap

PIXCL provides simple image printing services with the [PrintBitmap](#) command. The dialog boxes that appear are part of the Windows Common Dialogs library. All of the supported bitmap formats can be printed with this command.

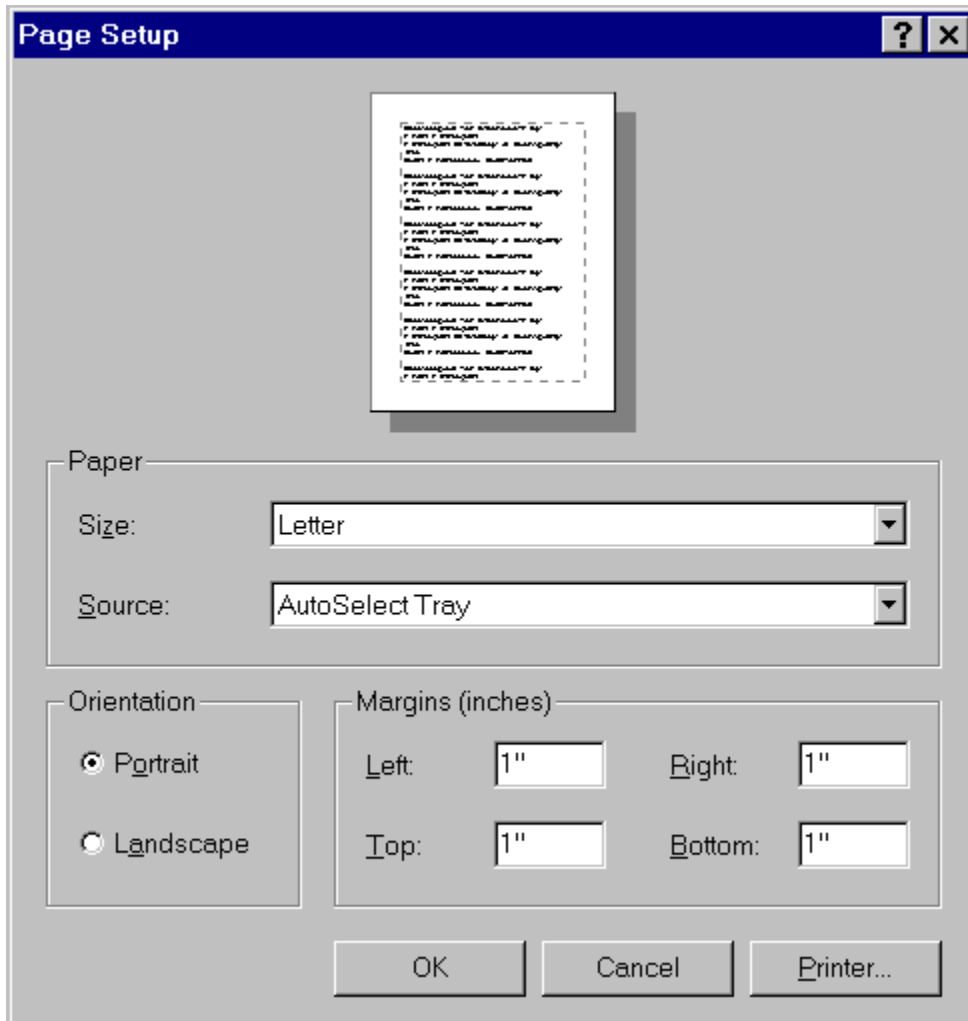
Syntax: [PrintBitmap](#)(*FileName\$,SETUP | PRINT_PTRRES | PRINT_PAGERES, Result*)

Parameters:

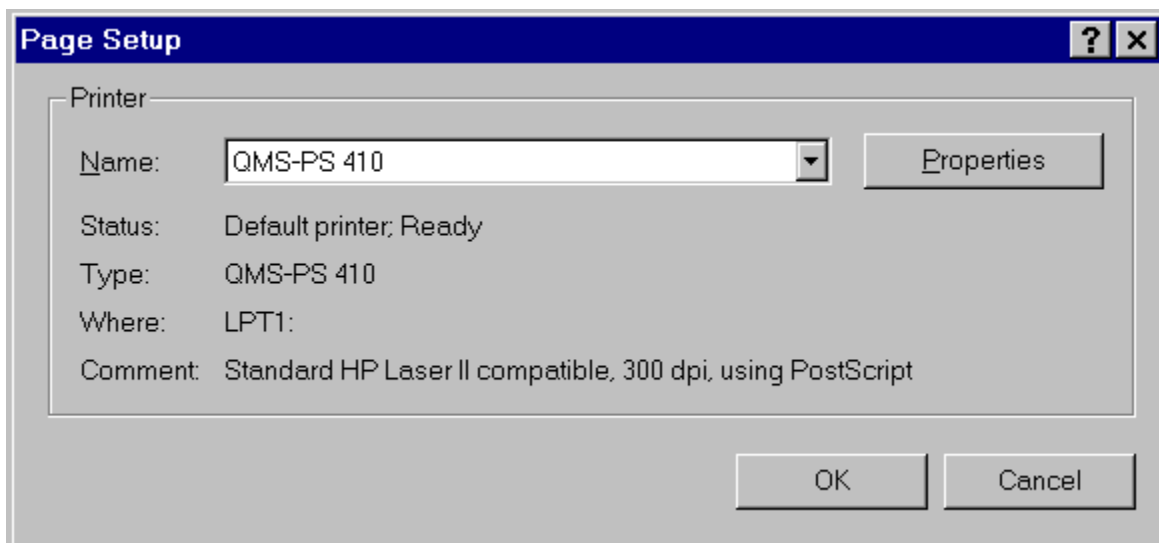
| | |
|--------------------------------|---|
| FileName\$ | The file to be printed. Bitmap pixels are assumed to be square. |
| SETUP | Displays the PrintSetup common dialogs, shown below. FileName\$ is ignored and can be a NULL string. |
| PRINT_PTRRES | Displays the Print dialog, shown below, and starts the job. The bitmap is scaled from the PRINTER resolution to the printer page resolution, and centered on the page. If the bitmap width is smaller than the number pixel per printer page, there will usually be a large unprinted area with this mode, while retaining the aspect ratio. If the image is bigger than the page resolution, only the center of the bitmap that fits will be printed. You would generally use this mode to check a bitmap output colors. |
| DPRINT_PTRRES | Passes the selected file directly to the current default printer. |
| PRINT_PAGERES | Displays the Print dialog, shown below, and starts the job. The bitmap is scaled from the BITMAP resolution to the printer page resolution, and centered on the page. This option is effect scales the bitmap to fit the page, while retaining the aspect ratio. |
| DPRINT_PAGERES | Passes the selected file directly to the current default printer. |
| Result | 1 if either the print setup or start print operation is successful,otherwise 0. |

Common Dialogs:

The standard dialogs for printer set up are shown below. The Page Setup dialog appears when you use the SETUP token in a PrintFile command.

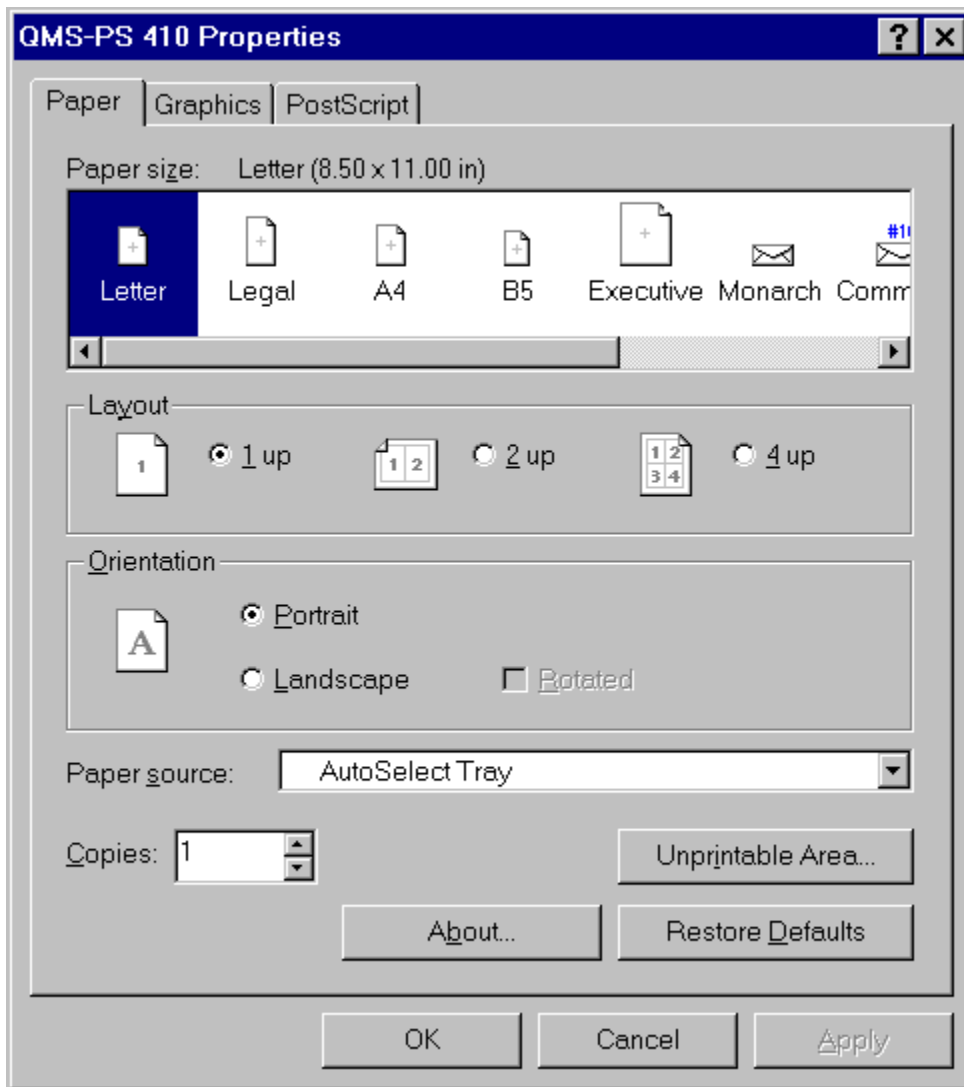


Pressing OK will save the setup parameters in the PIXCL application. Pressing Cancel will close the Page Setup dialog. Pressing the Printer button in the above dialog will display the following secondary dialog box.

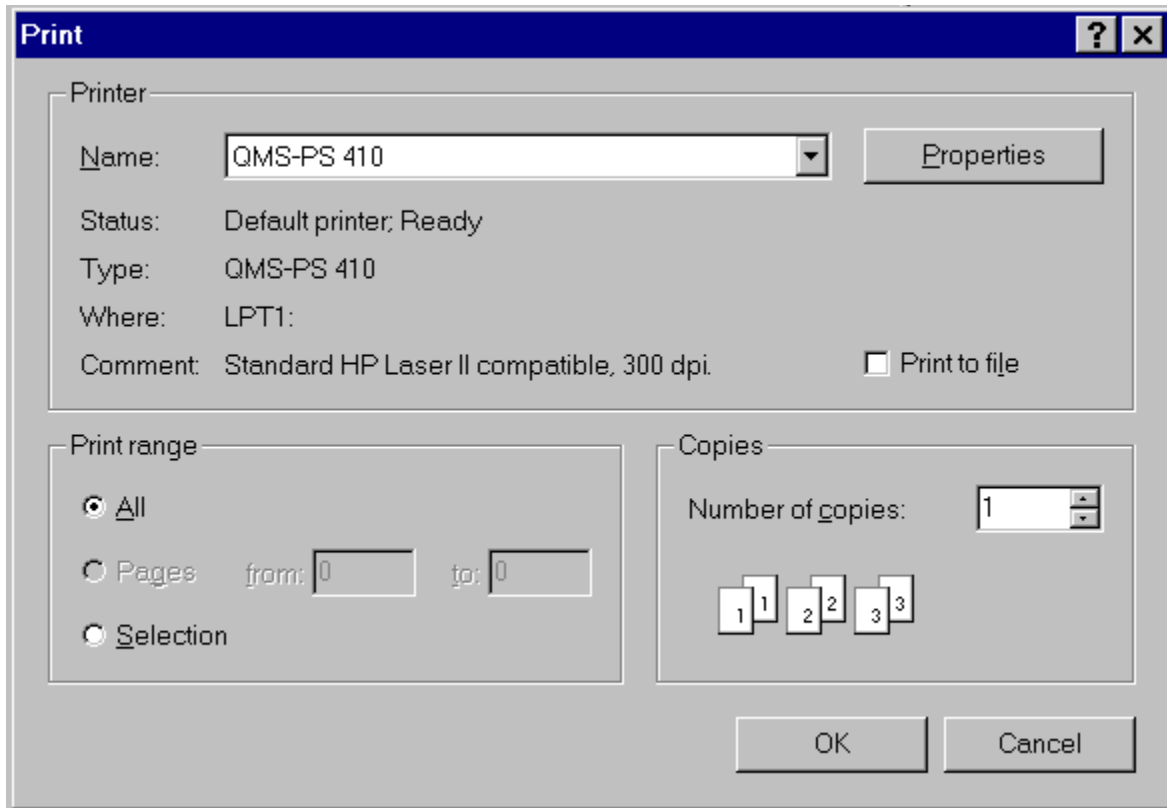


The Name combobox above shows the available printers. There may be more than one printer available on your system, and there

may be network available printers as well. Pressing the Properties button in the above dialog will display the following:



When you use the PRINT token in a [PrintBitmap](#) command, the standard dialog for printing a document, below, is displayed. Pressing the OK button will submit the print job. Pressing the Cancel button will close the Print dialog.



Remarks:

A common script programming error is to assign a string variable the name `PrintFile$`. This will result in a syntax error because the variable name is the same as the command name string. PIXCL requires that variables not have the same name string as any reserved command or keyword.

Once a SETUP token version of `PrintBitmap` has been issued and completed successfully, the selected printer settings are retained until the another SETUP command is issued, or the PIXCL application exits.

Using either of the two `PRINT` tokens displays the Print dialog, and if you click OK, starts the print job. The `DPRINT` tokens start the print job immediately on the default printer.

If you have a network printer (i.e. connected directly on to the network) such as an HP IIISi, you will need to identify the printer by its network port number for the print job to be successfully submitted.

Example:

```
PrintFileDialog:
    DrawBackground
    FileToPrint$ = "i:\v196beta\v_image.ini"
    FileExist (FileToPrint$,Res)

If Res = 1
    PrintBitmap(FileToPrint$,SETUP,Res)
    If Res = 0 Then DrawText(10,10,"Failed to start print.")
Else
    MessageBox (OK,1,EXCLAMATION,
"File to be printed cannot found.",FileToPrint$,Res)
```

```
Endif  
    PrintBitmap(FileToPrint$,PRINT_PAGERES,Res)  
    Goto Wait_for_Input
```

Related Commands:

[Run](#) [PrintFile](#) [EnumPrinters](#)

PrintFile

When you need to print a document file from another application, it can be tedious to have to start the application, load the document, and print it. Fortunately, most applications have a way of printing documents in the background, using a Windows function called the shell.

The shell provides, amongst other functions, the ability to print files. When the [PrintFile](#) command is processed, the shell looks at the file type and checks in the Windows registry for the application that is able to print the file. If you look in the Registry for a particular application, say MS-Word, you will see the entries `.../shell/open` and `.../shell/print`.

The shell starts the application in the background, loads the file to be printed and submits the job to the Windows print spooler. Once the job has been submitted successfully, the application is terminated. All you should see is a dialog box informing you that the job is printing. If the press the Cancel button in this dialog, the print job is stopped.

Please note: It is NOT PIXCL that is printing the file, it is the application associated with that file type.

Syntax: [PrintFile](#)(*FileToPrint\$,Result*)

Parameters:

FileToPrint\$ The full path and filename of the file you want to print.
Result 1 if the operation was successful, otherwise 0.

Example:

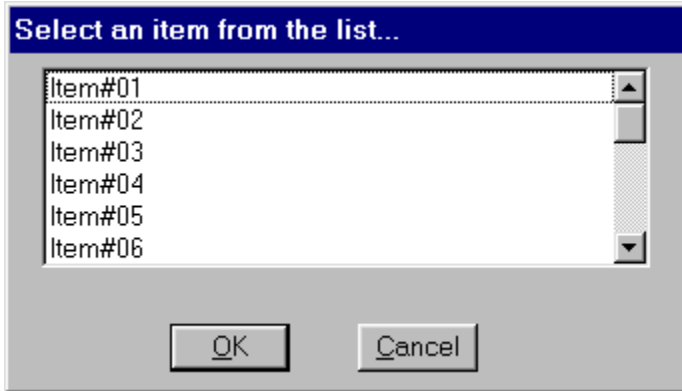
```
Printing_A_File:
    DrawBackground
    FileToPrint$ = "i:\v196beta\v_image.ini"
    FileExist (FileToPrint$,Res)
    If Res = 1
        PrintFile (FileToPrint$,Res)
        If Res = 0 Then DrawText(10,10,"Failed to print with the Shell.")
    Else
        MessageBox (OK,1,EXCLAMATION,
            "File to be printed cannot found.",FileToPrint$,Res)
    Endif
    Goto Wait_for_Input
```

Related Commands:

[PrintBitmap](#) [Run](#) [EnumPrinters](#)

ProgressBar

Draws a standard Windows progress bar, either at the bottom of the PiXCL application client area, or at specified client co-ordinates. Only one progress bar can be defined at one time. Progress bars are designed to indicate the state of any length process. The image below shows an example of a progress bar and a status bar.



Syntax: `ProgressBar(ENABLE|DISABLE,x1,y1,x2,y2)`

Parameters:

`ENABLE|DISABLE` Enable or disable the status bar child window..

`x1,y1,x2,y2` The client area co-ordinates that the progress bar is to be drawn. If all four of these co-ordinates are set to zero, the progress bar is drawn at the bottom of the client area, the same as is a status bar. If the progress bar is being disabled, these values are ignored.

Remarks:

If a status bar has been enabled, it takes precedence over a progress bar in the same location. That is, if a pane in a status bar is updated, a progress bar will be overwritten. It is possible to have a a progress bar displayed and within a status bar pane, if the status bar is not updated while the progress bar is displayed.

Example:

This program fragment draws the client area in the example image shown above.

EnableCustom:

```
UseBackGround(TRANSPARENT,192,192,0)
DrawBackGround
ProgressBar(ENABLE,5,100,200,120)
DrawTextExt(10,10,300,90,
"Example of a partially filled progress bar, with a status bar as well.",LEFT)
StatusWindow(ENABLE,BOTTOM,1,290,-1,0,0)
DrawStatusWinText(0,"Long process under way ... please be patient!")
Goto Wait_for_Input
```

Related Commands:

[UpdateProgressBar](#)

PulseEvent

Events may be used or required by extension libraries only. For more information, see the topics on Synchronization Functions in your compiler documentation. The **PulseEvent** function provides a single operation that sets (to signaled) the state of the specified event object and then resets it (to nonsignaled) after releasing the appropriate number of waiting threads.

Syntax: [PulseEvent\(EventHandle,Result\)](#)

Parameters:

[EventHandle](#) The event handle created by [CreateEvent](#).
[Result](#) 1 if the function succeeds, otherwise 0.

Related Commands:

[CreateEvent](#) [CloseEvent](#) [ResetEvent](#)

PXLCmds

This command returns the current number of fixed and variable argument commands. [PXLCmds](#) is normally only used when you need to verify the version of PiXCL you are using.

Syntax: [PXLCmds](#)(*FixedArgCmds*, *VariableArgCmds*)

Related Commands:

None

PXLResume, PXLResumeAt

These commands are designed to be used with the `WaitInput()` command, and sends a message to a PiXCL application to continue interpreting of the script at the current point or at a specific label. For example, if you want a PiXCL application to launch another application, and wait until the second application is completed, you would use a `WaitInput()` command (Note no arguments). You can of course use a polling method such as checking if a window exists or not, or polling the Clipboard for a particular text string, but these all take up processor cycles, and does not make use of the message passing nature of Windows.

It is not appropriate to use `PXLResume` with a `WaitInput(n)` command, as the PiXCL application will resume interpreting once the wait-time is complete.

With `PXLResumeAt`, you can create a variety of separate PiXCL applications that can operate independently when required, but also have other PiXCL applications react to explicit events that occur outside of their process space. For example, you can have one PiXCL application processing some images, and send another PiXCL application a message to update a progress bar display.

If you are programming extension or fast processing functions in another language, the message passing operation of `PXLResumeAt` can also be simply included in your code. See the Remarks section below.

Syntax: `PXLResume(WindowName$, Result)`

`PXLResumeAt(`

`To_WindowName$,Label$,From_WindowName$,Result)`

Parameters:

`WindowName$` The name of the target window. This must be a PiXCL application. You must ENSURE that the exact window name is used (case insensitive) or the function will fail.
`FromWindowName$` can be a null string if desired.

`Label$` A specific label in the target PiXCL application. Note this has to be a string or string variable in this command, as it is passed to the target application.

`Result` 0 if the operation is not successful, the message cannot be posted or the target window does not exist. If you use a NULL string, the command is ignored, but the result is still 1, as the command has successfully done nothing...

Remarks:

Sending these messages to any window other than a PiXCL application parent window with either have no effect (i.e. it will be ignored by the target application), or can have an unpredictable effect.

Sending a resume message to itself or to a NULL windowname has no effect.

If you are writing additional programs with a C or C++ compiler to interact with a PiXCL 4.1 or later program, the resume message should be declared as

```
#define WM_USER_INTERPRETSCRIPT (WM_USER+50)
```

```
#define WM_USER_INTERPRETSCRIPTAT (WM_USER+53)
```

`Label$` must be concatenated with `From_Windowname$` with a space delimiter to form a text string that is passed to the clipboard owned by the target application window.

Example:

See the sample files **comms1.pxI** and **comms2.pxI** for examples of how this command can be used.

Related Commands:

[WaitInput](#), [EnumWindows](#) [GetCopyDataMsg](#) [SendCopyDataMsg](#)

QueryRecycleBin

For systems with Internet Explorer 4.01 installed, it is possible to query the contents of the Recycle Bin.

Syntax: [QueryRecycleBin](#) (*Drive\$,Size,Items*)

Parameters:

| | |
|-------------------------|---|
| Drive\$ | The drive on which the Recycle Bin(s) are located. If an empty string "" is used, all Recycle Bins on all drives are queried. |
| Size | The size in KB of the Recycle Bin(s). |
| Items | The number of items in the Recycle Bin(s). |

Remarks:

This command requires **shell32.dll** version 4.72 or later.

Related Commands:

[DrawShellIcon](#) [EmptyRecycleBin](#)

Random

Returns a pseudo-random positive integer number within a specified range. This command is useful for generating random colors for pens and brushes, or co-ordinates for graphic draw commands.

Syntax: `Random(Range,RandomNumber)`

Parameters:

`Range` A number between 0 and 32767.

`RandomNumber` The random number returned from the generator..

Remarks

The pseudo-random number generator generates a number from a seed value (the system time the first time the command is issued) in the range 0 to 32767. Hence, for any particular program that uses the Random command, the first number in the sequence will always be different. The range value is used to take the modulus of the generated number before it is passed back to your PiXCL script. For example, a random number of 723, and a `Range` of 255 (i.e. $723 \bmod 255$) returns 213.

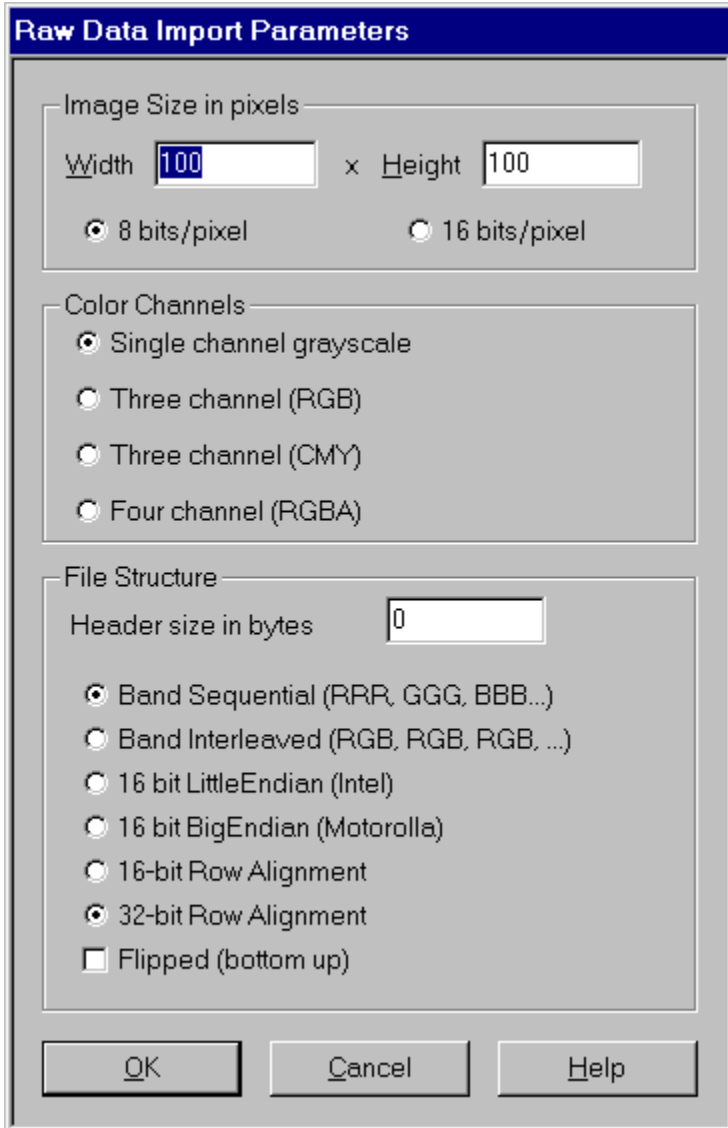
Example:

```
Random(1000,Number)
```

```
Random(255,Color)
```

RawDataParamBox

When working with raw bitmap data, it is often useful to be able to enter the image parameters via a dialog box. An example is shown below. The return values from this command are passed directly to the [ReadRawBitmap](#) command. If you are working with raw data of the same parameters on a regular basis, you could store the parameters in an INI style file that is read to initialize the arguments for [ReadRawBitmap](#), rather than using [RawDataParamBox](#).



Syntax: [RawDataParamBox](#)(*RawImage\$,Help\$,Xsize,Ysize,Samples,Bits,Offset,Flags*)

Parameters:

- [ImageData\\$](#) The raw image data file on disk. The default extension is **.raw**, but can be anything required by your application. This is the filename that appears in the PiXCL image list with the [ListLoadedBitmaps](#) command. If [ImageData\\$](#) is not null, it replaces the title bar text shown above.
- [Help\\$](#) Optional Help string that is displayed in a MessageBox when the Help button in the dialog is pressed. Can be set to a null string if desired.
- [Xsize,Ysize](#) The returned number of pixels per line, and number of lines of the data. If the variables have been previously created, the current values appear in the edit controls.
- [Samples](#) The returned number of samples per pixel.

| | |
|---------------|--|
| <i>Bits</i> | The returned number of bits per sample. Can be 1, 4, 5, 8 or 16. |
| <i>Offset</i> | The returned offset from the beginning of the file to the start of the image data. This is used to skip binary header information. If the variable has been previously created, the current value appears in the edit control. |
| <i>Flags</i> | Returned data formatting flags. This value is obtained from the RawDataparamBox command. <i>Flags</i> is also the return variable. If the operation is successful, returns 1, otherwise is set to 0. |

Remarks:

Xsize, Ysize, Samples, Bits, Offset are variables that can be used to initialize the edit fields in the dialog. Default values are shown in the sampel dialog above. *Flags* is binary combination of the bits per pixel, color channel and file structure settings returned from the dialog.

Related Commands:

[FileRead_Binary](#) [ReadRawBitmap](#)

RDBCcloseKey

The RDBCcloseKey function releases the handle of the specified key. All keys should be closed once they no longer need to be accessed.

Syntax: [RDBCcloseKey\(InHandle, Result\)](#)

Parameters:

[InHandle](#) A valid open handle or predefined handle. Permanently open keys cannot be closed, and if used, Result returns 0.

[Result](#) 0 if the operation failed, otherwise 1.

Remarks

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCreateKey](#), [RDBDeleteKey](#), [RDBEnumKey](#), [RDBOpenKey](#), [RDBQueryKey](#), [RDBQueryValue](#), [RDBSetValue](#)

RDBCreateKey

The [RDBCreateKey](#) function creates the specified key. If the key already exists in the registry, the function opens it.

Syntax: [RDBCreateKey\(InHandle, SubKey\\$, ObjectType\\$, OutHandle, Result\)](#)

Parameters:

| | |
|------------------------------|---|
| InHandle | A valid open handle or predefined handle. Permanently open keys cannot be closed, and if used, Result returns 0. |
| SubKey\$ | The string specifies the name of a subkey that this function opens or creates. SubKey\$ must be a subkey of the key identified by the InHandle parameter. This subkey must not begin with the backslash character ('\'). This parameter cannot be NULL. |
| ObjectType\$ | Specifies the class (object type) of this key. This parameter is ignored if the key already exists. This parameter is related to OLE, and can be set to a NULL string. |
| OutHandle | A valid open handle, otherwise 0 or undefined. |
| Result | 0 if the operation failed, otherwise 1. |

Remarks

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCcloseKey](#), [RDBDeleteKey](#), [RDBEnumKey](#), [RDBOpenKey](#), [RDBQueryKey](#), [RDBQueryValue](#), [RDBSetValue](#)

RDBDeleteKey

There are some variations between the result of this function in Windows 95 and Windows NT 3.51.

Windows 95: The [RDBDeleteKey](#) function deletes a key and all its descendents.

Windows NT: The [RDBDeleteKey](#) function deletes the specified key. This function cannot delete a key that has subkeys, so it is necessary to delete the last key in each branch at a time.

Syntax: [RDBDeleteKey](#)(*InHandle*, *SubKey\$*, *Result*)

Parameters:

| | |
|--------------------------|--|
| InHandle | A valid open handle or predefined handle. Permanently open keys cannot be closed, and if used, Result returns 0. |
| SubKey\$ | The string specifies the name of a subkey that this function deletes. SubKey\$ must be a subkey of the key identified by the InHandle parameter. This subkey must not begin with the backslash character ('\'). This parameter cannot be NULL. |
| Result | 0 if the operation failed, otherwise 1. |

Examples:

This will delete a previously created subkey, because its root is [@RDB_CLASSES_ROOT](#)

```
RDBDeleteKey(@RDB_CLASSES_ROOT, "PiXCL4.2", Res)
```

whereas if you get a Handle with the (say) [RDBOpenKey](#) command, like this,

```
RDBOpenKey(@RDB_CLASSES_ROOT, "PiXCL4.2", Handle)
RDBDeleteKey(Handle, "PiXCL4.2", Res) {fails,
    SubKey$ is not subkey of handle}
```

"PiXCL4.2" is not a subkey of [Handle](#) (it is the string pointed to by Handle)

Remarks

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCcloseKey](#), [RDBCreateKey](#), [RDBEnumKey](#), [RDBOpenKey](#), [RDBQueryKey](#), [RDBQueryValue](#), [RDBSetValue](#)

RDBEnumKey

[RDBEnumKey](#) enumerates subkeys of the specified open registry key [InHandle](#). The function retrieves information about one subkey each time it is called.

Syntax: [RDBEnumKey\(InHandle,Index,SubKeyName\\$,ClassName\\$,Result\)](#)

Parameters:

| | |
|------------------------------|--|
| InHandle | A valid open handle or predefined handle. Permanently open keys cannot be closed, and if used, Result returns 0. |
| Index | Specifies the index of the subkey to retrieve. This parameter should be zero for the first call to the RDBEnumKey function and then incremented for subsequent calls. Because subkeys are not ordered in the Registry, any new subkey will have an arbitrary index. This means that the function may return subkeys in any order. You can also use the RDBQueryKey command to get the maximum number of keys for Index , then decrement to zero. |
| SubKeyName\$ | The returned string of the subkey name at the specified index. If SubKeyName\$ returns a NULL string, no more subkeys can be located at InHandle . Returns a NULL string if Result = 2. |
| ClassName\$ | Returned string containing the name of the class. Relevant for OLE operations. . Returns a NULL string if Result = 2. May return a NULL anyway. |
| Result | 0 if the operation failed, 1 if the operation was successful, and 2 if the operation was successful but found no additional subkeys to enumerate. |

Remarks

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCcloseKey](#), [RDBcreateKey](#), [RDBdeleteKey](#), [RDBopenKey](#), [RDBqueryKey](#), [RDBqueryValue](#), [RDBsetValue](#)

RDBOpenKey

Opens a Registry subkey of the specified handle or predefined handle.

Syntax: [RDBOpenKey\(InHandle,SubKey\\$, OutHandle\)](#)

Parameters:

[InHandle](#)

A valid open handle or predefined handle. Permanently open keys cannot be closed, and if used, [Result](#) returns 0.

[SubKey\\$](#)

The name of the application or known subkey. For example, if we want to access the Registry entries for MS-Paint, [SubKey\\$](#) requires a value of "PBrush". Keys are not case sensitive. The [SubKey\\$](#) specified must be a subkey of the key identified by the [InHandle](#) parameter. [SubKey\\$](#) must not begin with the backslash character ("\").

If this parameter is a NULL i.e. an empty string, the function will open a new handle of the key identified by the [InHandle](#) parameter. In this case, the function will not close the handles previously opened.

[OutHandle](#)

0 if the operation failed, or non-zero (and usually large negative) if the operation succeeded.

Remarks

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCcloseKey](#), [RDBCreateKey](#), [RDBDeleteKey](#), [RDBEnumKey](#), [RDBQueryKey](#), [RDBQueryValue](#), [RDBSetValue](#)

RDBQueryKey

This Registry command returns an assortment of information about a specified key or predefined constant.

Syntax: [RDBQueryKey](#)(
[InHandle](#), [ClassName\\$](#), [NumberOfSubKeys](#), [NumberOfValues](#), [Result](#))

Parameters:

[InHandle](#) A valid open handle or predefined handle. Permanently open keys cannot be closed, and if used, [Result](#) returns 0.

[ClassName\\$](#) Name string related to OLE operations. Can be NULL.

[NumberOfSubKeys](#) Number of subkeys under InHandle.

[NumberOfValues](#) Number of values under InHandle. Subkeys can have multiple values.

[Result](#) 0 if the operation failed, 1 if the operation was successful.

Remarks

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCcloseKey](#), [RDBcreateKey](#), [RDBdeleteKey](#), [RDBenumKey](#), [RDBqueryValue](#), [RDBsetValue](#)

RDBQueryValue

Use this command to retrieve the Registry subkey value as a string. This string can be either a string, '|' delimited string or a 32 bit binary value expressed as an integer.

Syntax: [RDBQueryValue\(InHandle, SubKey\\$, SubKeyRtn\\$, Result\)](#)

Parameters:

| | |
|-----------------------------|---|
| InHandle | An integer value obtained from an RDBOpenKey , or one of the predefined handles. |
| SubKey\$ | The name of the value to query. If a value with this name is not already present in the key, the function returns 0 in Result . SubKeyRtn\$ = "" is a possible successful return. |
| SubKeyRtn\$ | The string representation of the key. This may be a STRING, a ' ' delimited string LIST, or a DWORD integer string. If you need the integer string converted to a numeric value, use the Val command. |
| Result | 0 if the operation failed, or 1 if successful. |

Remarks

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCcloseKey](#), [RDBCreateKey](#), [RDBDeleteKey](#), [RDBEnumKey](#), [RDBOpenKey](#), [RDBQueryKey](#), [RDBSetValue](#)

RDBSetValue

The RDBSetValue function stores data in the value field of an open registry key. It can also set additional value and type information for the specified key.

Syntax: `RDBSetValue(InHandle,SubKey$,Value$, TOKEN, Result)`

Parameters:

| | |
|-----------------|--|
| <i>InHandle</i> | An integer value obtained from an RDBOpenKey , or one of the predefined handles. |
| <i>SubKey\$</i> | The name of the value to set. If a value with this name is not already present in the key, the function adds it to the key. Set this to NULL to write a <i>Value\$</i> into the (Default) subkey. |
| <i>Value\$</i> | The string representation of the value to be stored. Data can be strings, delimited lists, integers or binary data, according to the token specified. |

Available TOKENs are

| | |
|---------------|--|
| STRING | Any string variable or string literal. |
| LIST | A delimited list. The delimiter must be the " " (pipe) character, and the last string must have the " " following. In the write to the Registry, these " " are converted to NULLs, and the whole list has two NULLs following. |
| DWORD | Any number. Use the Str command to convert any numerical value (a 32 bit integer can represent any 32 bit floating point or hex number) to a string. |
| BINARY | A delimited set of integer numbers, such as window placement coordinates. The delimiter must be the " " (pipe) character. The last string does NOT require a following delimiter. Number strings are converted into binary and stored in the specified Registry key. |
| <i>Result</i> | 0 if operation fails, else 1. |

Remarks:

Value lengths are limited by available memory. Long values (more than 2048 bytes) should be stored as files with the filenames stored in the registry, as this helps it perform efficiently. Application elements such as icons, bitmaps, and executable files should be stored as files and not be placed in the registry.

STRINGs appear in quotes in a Registry entry e.g. "h:\app\pixcl.exe,1".

LISTs appear in a Registry entry as a sequence of ASCII character in hexadecimal notation. e.g. 42 4d 50 00 52 4c 45 00 00 is the list entered as "BMP|RLE|"

DWORDs are 32 bit numbers and appear in a Registry entry as Hexadecimal strings plus the decimal equivalent e.g. 0x00000100 (256)

Example:

See sample file [registry.pxl](#) for a working example of all registry commands.

Related Commands:

[RDBCcloseKey](#) [RDBCcreateKey](#) [RDBDeleteKey](#) [RDBEnumKey](#) [RDBOpenKey](#) [RDBQueryKey](#) [RDBQueryValue](#)

ReadBitmapID

PIXCL 5 command. Any bitmap loaded into the PIXCL image list can have an identifier string written into and read from the bitmap data at an arbitrary location. Because strings are ascii characters in the range 0-127 and 128-255 for other characters, an identifier string will often be in effect invisible. The general term for this hiding of data within other data is steganography. It can be handy to identify images for which you hold the copyright.

Syntax: [ReadBitmapID\(Imagename\\$, Pixel, Line, BytesToRead, Idstring\\$, Result\)](#)

Parameters:

| | |
|-----------------------------|---|
| Imagename\$ | The image loaded into the PIXCL image list. If the image is not loaded, the function fails. |
| Pixel, Line | The start coordinate for the ID string. |
| BytesToRead | The number of bytes to read. |
| Idstring\$ | The string read from the image. |
| Result | 1 if the operation succeeded, otherwise 0. |

Related Command:

[WriteBitmapID](#) [SaveBitmap](#)

ReadBitmapRect

Reads a sub-area of a BMP or TIF bitmap on the disk, into the current image in the PiXCL image list. The TIF bitmap can be compressed (except for LZW mode, which is not supported) or uncompressed. Reading subareas of other formats is not supported in the current version of PiXCL.

Syntax: [ReadBitmapRect\(ImageFile\\$,x1,y1,x2,y2,Result\)](#)

Parameters:

| | |
|-----------------------------|--|
| ImageFile\$ | The target image on the disk. This is the name that is written to the image list when the image is loaded. |
| x1,y1,x2,y2 | The region in the target bitmap on disk that is to be read into the PiXCL list. |
| Result | 1 if the operation is successful, otherwise 0. |

Related Commands:

[WriteBitmapRect](#)

ReadCommPort

Asynchronous data streams typically include variable length data, followed by a terminating character or string. Most useful serial devices like digitizing tablets let you configure the terminating string. In the current version of PiXCL, this string MUST be set to a carriage return and linefeed pair (CR-LF).

PiXCL maintains a read buffer of about 4 KB. Most input will be far less than this size. When the CR-LF is received, a comms event is generated to which your PiXCL program can respond, in much the same way that your program responds to mouse or keyboard events in the [WaitInput\(\)](#) idle loop.

Hence, to receive a read or write comms event, you would use a command [WaitCommsEvent\(R,<label>,Timeout\)](#), and at [<label>](#), you would use a [ReadCommPort](#) command to get the current buffer.

Syntax: [ReadCommPort\(COMx,Data\\$\)](#)

Parameters:

[COMx](#) Port, where x = 1 – 4 for standard PCs. **PIXCL 5:** If you have a multiple COM port board installed, ports 5-13 are supported.

[Data\\$](#) The data read in from the port buffer

Related Commands:

[EscCommFunction](#) [ClearCommPort](#) [GetCommPort](#) [SetCommPort](#) [WaitCommEvent](#) [WriteCommPort](#)

ReadConsole

PIXCL 5 Command. Read the current line from a console window.

Syntax: [ReadConsole\(Command\\$\)](#)

Parameters:

Related Commands:

[ShowConsole](#) [FreeConsole](#) [WriteConsole](#)

ReadRawBitmap

You can read raw bitmap data array files into a bitmap that is loaded into the PiXCL image list. Once in the list in memory, you can apply image processing operations and save the bitmap to one of the supported bitmap formats.

Syntax: [ReadRawBitmap\(ImageData\\$,Xsize,Ysize,Samples,Bits,Offset,Flags\)](#)

Parameters:

| | |
|-----------------------------|---|
| ImageData\$ | The raw image data file on disk. The default extension is .raw , but can be anything required by your application. This is the filename that appears in the PiXCL image list with the ListLoadedBitmaps command. |
| Xsize,Ysize | The number of pixels per line, and number of lines of the data. |
| Samples | The number of samples per pixel. |
| Bits | The number of bits per sample. Can be 1, 4, 5, 8 or 16. |
| Offset | The offset from the beginning of the file to the start of the image data. This is used to skip binary header information. |
| Flags | Integer VARIABLE. Data formatting flags. This value is obtained from the RawDataParamBox command. If the operation is successful, returns a non-zero image handle identifier, otherwise is set to 0 on exit. |

Related Commands:

[RawDataParamBox](#) [ListLoadedBitmaps](#)

ReadTIFcompressMode

PIXCL supports several compression modes in TIFF files, with the exception of LZW (because of the copyright licensing issues). This command reads the file header, and can be used to identify TIF images with LZW compression, as an error check. This command will also return the compression mode of other file types (e.g. JPEG) if relevant.

Syntax: [ReadTIFcompressMode\(TIFimage\\$, CompressMode\)](#)

Parameters:

[TIFimage\\$](#) The image whose header is read to get the compression mode. If the image file does not exist, [CompressMode](#) returns 0.

[CompressMode](#)

| | |
|----------------|---|
| 1 = NONE | No compression. |
| 2 = LZW | LZW compression (not supported DrawBitmap functions). |
| 3 = PACKBITS | PackBits compression. |
| 5 = CCITT_FAX3 | Group 3 fax. Relevant to monochrome bitmaps only. |
| 6 = CCITT_FAX4 | Group 4 fax. Relevant to monochrome bitmaps only. |
| 8 = RLE | Run Length Encoded. |
| 9 = JPEG | JPEG compression. |
| 10 = DEFLATE | Deflate compression. |

Related Commands:

[SaveTIFcompressMode](#) [SaveBitmap](#) [SaveRectangle](#)

Redraw

Forces a redraw of the memory to screen bitmaps. [Redraw](#) takes no parameters. Use this command only when modifying the raster operation code with the [SetROPcode](#) command. Use anywhere else has no visual effect, though it does redraw the client area.

Related Commands:

[SetROPcode](#)

RegisterExtLibCmdSet

This command is an alternative to [RegisterUserCommand](#), and is designed to load all the commands in a command extension library at once.

[RegisterExtLibCmdSet](#) is generally used somewhere at the start of a program, but can be anywhere before the user commands are to be parsed and executed. User commands are implemented in DLL's, with multiple commands within a DLL possible. The DLL is loaded to access the command load function that is written in the DLL, then usually, but not always, unloaded.

The User Command Library API is an available option for PiXCL v4.4 and later, and ships with the geoPiXCL product.

In summary however, an extension command DLL can include new image processing algorithms that access the (geo)PiXCL image list and (geo)PiXCL internal structures directly, and also custom dialogs and other Windows resources. User command syntax is defined within an extension command DLL, and are normally documented with an extension HLP file that comes with the DLL. This HLP file can be linked into the PiXCL MDI editor. VYSOR Integration provides an extension command development service.

Syntax: [RegisterExtLibCmdSet\(DLLName\\$,LOADDLLNOW | LOADONDEMAND, Result\)](#)

Parameters:

| | |
|------------------------------|---|
| DLLName\$ | The name of the DLL that contains the extension commands. DLLname\$ can be either just the name of the DLL without path or extension (which requires it to be in either the current directory, windows or system directory, or a directory in the PATH environment string), or the full path and extension. |
| LOADDLLNOW | This token instructs PiXCL to load the extension command DLL immediately, and leave it attached to the PiXCL process. |
| LOADONDEMAND | This token instructs PiXCL to load the DLL immediately. When the PiXCL application terminates, the DLL is automatically unloaded. The UnregisterUserCmd command can also be used to unload a DLL. |
| Result | The number of commands successfully registered, otherwise 0. |

Related Commands:

[RegisterUserCommand](#) [UnregisterUserCmds](#)

RegisterUserCommand

All user written extension commands have to be registered with a PiXCL and geoPiXCL application before they can be used, or else a syntax error will occur. Registering any command automatically enables user command processing.

`RegisterUserCommand` is generally used somewhere at the start of a program, but can be anywhere before the user command is to be parsed and executed. User commands are implemented in DLL's, with multiple commands within a DLL possible.

The User Command Library API is an available option for PiXCL v5.0 and later, and ships with the geoPiXCL product.

In summary however, an extension command DLL can include new image processing algorithms that access the (geo)PiXCL image list and (geo)PiXCL internal structures directly, and also custom dialogs and other Windows resources.

Syntax: `RegisterUserCommand(DLLName$, CmdName$, Result, arg_type#1, . . . , arg_type#n, P_NUM_VARIABLE)`

Parameters:

`DLLName$`

The name of the DLL that contains `CmdName$`. `DLLName$` can be either just the name of the DLL without path or extension (which requires it to be in either the current directory, windows or system directory, or a directory in the PATH environment string), or the full path and extension.

`CmdName$`

The string that is user for the command e.g. "UserCommand1". Commands have the same constraints as built in commands: must start with "a-z". This command name must not be the same as any predefined command or keyword, or a syntax error will occur.

`Result`

1 if the command was successfully registered, 2 if the command is already registered, otherwise 0.

`arg_type#n`

The following argument types are defined. Note that some of these take two or three argument buffer entries. The maximum number of buffer entries is 64.

| token_value | meaning |
|-------------------------------------|---|
| <code>P_NUMBER</code> | static integer |
| <code>P_NUM_VARIABLE</code> | integer variable |
| <code>P_FP_NUMBER</code> | static float |
| <code>P_FP_VARIABLE</code> | float variable |
| <code>P_STRING</code> | static string |
| <code>P_STR_VARIABLE</code> | string variable |
| <code>P_COORDINATE</code> | integer pair |
| <code>P_RGB</code> | integer triplet (0-255) |
| For PiXCL 5.0 and later only | |
| <code>P_RGBA</code> | integer quad (0-255) |
| <code>P_RECTANGLE</code> | integer quad |
| <code>P_SQ_BRACKET_LEFT</code> | A [used with array variables |
| <code>P_SQ_BRACKET_RIGHT</code> | A] used with array variables |
| <code>P_VARIANT</code> | Indicates an integer, float or string can be used. |
| <code>P_VARIANT_VARIABLE</code> | Indicates an integer, float or string variable can be used. |
| <code>P_VARIANT_ARRAY</code> | Indicates an array variable, and must be followed by <code>P_SQ_BRACKET_LEFT</code> , <code>P_NUMBER</code> , <code>P_SQ_BRACKET_RIGHT</code> , |

While the number of arguments is arbitrary, some argument types and position are mandatory. Insertion of left and right brackets and comma delimiters are automatic.

1. The last argument **MUST** be a `P_NUM_VARIABLE`, as this is the default error return variable for the user command. A

syntax error will result if this is not correct. This variable also returns 0 if the User Command DLL can't be located, and -1 if the user command can't be located in the DLL.

2. Once registered, a command remains registered until the geoPiXCL application exits.
3. When the new command is executed, (geo)PiXCL loads the indicated library, performs the command, ythen unloads the library.
4. Commands must be registered in **STRICT** alphabetical order. Note that **Function_10** comes between **Function_1** and **Function_2**.
5. Array variables declared and called are slightly different because of the way the command parser works. For example a command `MyFunction(... P_VARIANT_ARRAY, P_SQ_BRACKET_LEFT, P_NUMBER, P_SQ_BRACKET_RIGHT, P_NUM_VARIABLE)` is correctly called by `MyFunction(MyArray[0], Result)`, rather than by `MyFunction(MyArray, [0], Result)` which is of course incorrect syntax.

It is not necessary to indicate the number of arguments, as this is calculated automatically.

A typical command might be

```
RegisterUserCommand(DLLname$,  
    "UserCmd1", Res, P_STRING, P_COORDINATE, P_NUM_VARIABLE)
```

This user command is later usable as, say,

```
UserCmd1(StringVar$, 12,70, Res)
```

Related Commands:

[CountRegdUserCmds](#) [RegisterExtLibCmdSet](#) [UnregisterUserCmds](#)

ReleaseCOM

PIXCL 5 command: A PiXCL application can become a COM client to a COM server. Calling this command initializes the Windows COM libraries. When using PiXCL as a COM client, you will need to have the necessary access or programming or scripting information on the server application. When it is time to shut down the COM access one or more servers, the [ReleaseCOM](#) command is used.

The call to [ReleaseCOM](#) uninitialized the Windows COM libraries.

Syntax: [ReleaseCOM\(\)](#)

Parameters:

None (for now)

Related Commands:

[CreateCOMinstance](#)

RemoveFont

The [RemoveFont](#) command removes the font resource from the Windows font table, so is no longer available to Windows applications.

Syntax: [RemoveFont](#)(*FontFilename*,\$,*Result*)

Parameters:

[FontFilename](#)\$ A valid font file filename. The filename may specify either a .FON font resource file, a .FNT raw bitmap font file, a .TTF raw TrueType file, or a .FOT TrueType resource file.

[Result](#) 1 if the operation succeeded, otherwise 0. If the font is not installed, [Result](#) also returns 0.

Related Commands:

[DrawText](#) [DrawNumber](#) [AddFont](#) [UseFont](#)

RenameListImage

When you want to rename an image loaded in the PiXCL image list, use the [RenameListImage](#) command. Changing the name of a loaded image will not change the name of the original file on the disk.

In **PIXCL 5** and **geoPIXCL**, JPG, PNG and TIF image format create Options fields in the image list record. Renaming also has to potentially change the image file type set when the original image was loaded from disk. If the image type is JPG, PNG or TIF, and the new type is different, any existing Options are deleted and if necessary the new empty Options recreated. Changing just the name not the extension has no effect on existing Options.

Syntax: [RenameListImage\(OldName\\$,NewName\\$,Result\)](#)

Parameters:

[OldName\\$](#) A current name in the image list.

[NewName\\$](#) The new list image name.

[Result](#) If [OldName\\$](#) does not exist in the list, or [NewName\\$](#) is a null string, [Result](#) returns 0. If the renaming is successful, [Result](#) returns 1.

Related Commands:

[ListLoadedBitmaps](#) [SetJPGOptions](#) [SetPNGOptions](#) [SetTIFFOptions](#)

ReportHistogramStats

Once a histogram or set of histograms has been displayed, some histogram statistics are available, using the [ReportHistogramStats](#) command.

Syntax: [ReportHistogramStats](#)(*HistogramID,Mean&, StdDeviation&,MinPixel,MaxPixel, ModePixel , ModeCount,AllStats\$*)

Parameters:

| | |
|------------------------------------|--|
| HistogramID | A histogram ID number returned by the Histogram command. |
| Mean& | The calculated mean value of the selected image channel. |
| StdDeviation& | The calculated standard deviation value of the selected image channel. |
| MinPixel, MaxPixel | The minimum and maximum pixel values from the histogram. |
| ModePixel | The pixel value with the greatest count. |
| ModeCount | The mode count. |
| AllStats\$ | All the previous statistics in a preformatted string. See Remarks below. |

Remarks:

The preformatted string for [AllStats\\$](#) is cr-lf delimited, and takes the form:

```
Mean : 126.77601
Standard Deviation : 26.90554
Minimum Pixel : 0
Maximum Pixel : 234
Mode Pixel : 118
```

[AllStats\\$](#) can be used directly in a messagebox or dialogbox.

Related Commands:

[Histogram](#) [ShowHistogram](#)

ReportMouse

This command reports the current coordinate position, and optionally the RGB values, in status window pane 0 as the mouse moves in the PiXCL application client area, or portion of the client area.

Syntax: `ReportMouse(x1,y1,x2,y2,xOffset,yOffset,xGain,yGain,
xZoom,yZoom,DISABLE|NORGB|RGB,STATUS|CHILD,x1,y1)`

Parameters:

| | |
|------------------------------|---|
| <code>x1,y1,x2,y2</code> | The rectangular region in which mouse reporting is active. This can be the whole client area if desired. |
| <code>xOffset,yOffset</code> | An offset value, used when you want pixel values for a zoomed or roamed displayed image. Default value is 0,0. Can be negative numbers. |
| <code>xGain,yGain</code> | A gain value. Normally 1,1, but can be used to invert mouse values. |
| <code>xZoom,yZoom</code> | Default value is 1,1. Used when a displayed image is zoomed. |
| <code>DISABLE</code> | Turn off mouse reporting. Other parameter values are ignored. |
| <code>NORGB</code> | Display mouse coordinates only in form XY: (x,y) |
| <code>RGB</code> | Display mouse coordinates and RGB values in form XY: (x,y), (r,g,b) |
| <code>STATUS</code> | Report the coordinates and RGB data in the status bar. |
| <code>CHILD</code> | Report the coordinates and RGB data in a floating window bar with top-left coordinates x1, y1. |
| <code>x1,y1</code> | The top left coordinates of the <code>CHILD</code> window, if created. These are ignored for the <code>STATUS</code> mode, and <code>DISABLE</code> mode. |

Remarks:

A status window must be enabled first with the `StatusWindow` command, or `ReportMouse` is ignored. If a region within the client area is defined, mouse reporting does not occur outside the defined area, and the status window pane 0 is cleared.

Example:

In this code fragment, a previously defined bitmap is loaded, and a reporting region the size of the displayed bitmap is specified. Reporting will occur independently of any other activity until the `ReportingOff` labeled code is executed.

```
ImageLoad:
  GetBitmapDim(Image$,Lines, Pixels, Bits)
  x2 += Pixels
  y2 += Lines
  DrawBitmap(20,20,Image$)
  StatusWindow(ENABLE,BOTTOM,1,200,0,0,0)
  ReportMouse(20,20,x2,y2,-20,-20,1,1,1,1,RGB)
  {
    or use this code to define a region ...
    WinGetClientRect("",cx1,cy1,cx2,cy2)
    ReportMouse(cx1,cy1,cx2,cy2,0,0,1,1,1,1,RGB)
  }
  Goto Wait_for_Input
  . . .
ReportingOff:
  ReportMouse(20,20,x2,y2,-20,-20,1,1,1,1,DISABLE)
  Goto Wait_for_Input
```

Related Commands:

[SetMouse](#), [StatusWindow](#), [DrawZoomedBitmap](#)

ResetEvent

Events may be used or required by extension libraries only. For more information, see the topics on Synchronization Functions in your compiler documentation. The **ResetEvent** function sets the state of the specified event object to nonsignaled.

Syntax: [ResetEvent\(EventHandle,Result\)](#)

Parameters:

[EventHandle](#) The event handle created by [CreateEvent](#).
[Result](#) 1 if the function succeeds, otherwise 0.

Related Commands:

[CreateEvent](#) [CloseEvent](#) [PulseEvent](#)

RestoreArray

PIXCL 5 command. The contents of array variables can be saved into binary files with the [SaveArray](#) command, and later restored with the [RestoreArray](#) command. The files are binary, with the the following format:

Bytes 0-3 = the number of entries in the array (current maximum 2048K);

Bytes 4-5 = the array type (1 = string, 3 = integer, 5 = float);

Bytes 6-7 = the number of arrays in the save file.

The array data follows: for integer and float arrays, each entry takes four bytes. For string arrays, each string entry is followed by two pipe characters i.e. "|". The last four bytes are a delimiter 0xffffffff, which is used only in multisave arrays. The contents of the file must match the size and type of the target array. A saved array of **n** elements can be restored to an array of **n+m** elements.

Syntax: [RestoreArray\(ArrayVariable,SaveFileName\\$,Result\)](#)

Parameters:

| | |
|--------------------------------|--|
| ArrayVariable | A pre-existing array variable, in the form Name[0] , Name\${0} or Name&[0] . |
| SaveFileName\$ | The path and name of the file from which the array is to be restored. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[Array](#) [FreeArrayVar](#) [FreeArrayVarAll](#) [SaveArray](#) [RestoreMultiArray](#)

RestoreMultiArray

PIXCL 5 command. The contents of multiple array variables can be saved into binary files with the [SaveMultiArray](#) command, and later restored with the [RestoreMultiArray](#) command. The files are binary, with the the following format:

Bytes 0-3 = the number of entries in the array (current maximum 2048K);

Bytes 4-5 = the array type (1 = string, 3 = integer, 5 = float);

Bytes 6-7 = the number of arrays in the save file.

The array data follows: for integer and float arrays, each entry takes four bytes. For string arrays, each string entry is followed by two pipe characters i.e. “|”. The last four bytes are a delimiter 0xffffffff. The contents of the file must match the size and type of the target array. A saved array of *n* elements can be restored to an array of *n+m* elements.

Syntax: [RestoreMultiArray\(ArrayVariable_1\[Size\],...,ArrayVariable_n\[Size\],SaveFileName\\$,Result\)](#)

Parameters:

[ArrayVariable\[Size\]](#) A pre-existing array variable, in the form [Name\[Size\]](#), [Name\\$\[Size\]](#) or [Name&\[Size\]](#). *Size* must be the size of the array as it was declared.

[SaveFileName\\$](#) The path and name of the file from which the array is to be restored.

[Result](#) The number of arrays restored if the operation was successful, otherwise 0.

Remarks:

The name of the array is NOT stored in the save file, so the target array has to be of the same type and at least the same size as the array record in the file. The order of the target arrays MUST match the order of the array records in the save file. If not, an error dialog will appear.

Related Commands:

[Array](#) [FreeArrayVar](#) [FreeArrayVarAll](#) [SaveArray](#) [SaveMultiArray](#)

Right

Returns a specified number of characters from the right of a string.

Syntax: `Right(String$,Places,Result$)`

Parameters:

String\$ The string from which you want to extract the text.

Places The number of places you want extracted.

Result\$ A string variable that will contain the result.

Example:

This program extracts the last word from the string " Ottawa Canada." and draws it at point (10,10) in the PIXCL window.

```
String$="Ottawa Canada"  
Len(String$,Length)  
Instr(String$," ",Location)  
Places = Length - Location  
Places = Places + 1  
Right(String$,Places,Result$)  
DrawText(10,10,Result$)  
WaitInput()
```

Related Commands:

[Right](#), [Instr](#), [Substr](#)

RightOf

Returns characters from the right of a location in a string.

Syntax: `RightOf(String$,Location,Result$)`

Parameters:

| | |
|-----------------|--|
| <i>String\$</i> | The string from which you want to extract the text. |
| <i>Location</i> | The location after which characters are copied to the <i>Result\$</i> string. The character at the specified location is not returned in <i>Result\$</i> . |
| <i>Result\$</i> | A string variable that will contain the result. |

Example:

This program extracts the last word from the string " Ottawa Canada." and draws it at point (10,10) in the PiXCL window.

```
String$="Ottawa Canada"  
Instr(String$," ",Location)  
RightOf(String$,Location,Result$)  
DrawText(10,10,Result$)  
WaitInput()
```

Related Commands:

[Right](#), [Instr](#), [Substr](#), [Left](#), [LeftOf](#), [StrRev](#)

RotateRectangle

This command lets you rotate blocks of pixels in a defined rectangle in a PIXCL application client area. Rotation can be vertical or horizontal, and the rate and number of iterations can be set. The effect is rather like scrolling an image up or down within a window. One possible use is to create a banner image and text, and scroll it across the screen or within a screen region.

Syntax: `RotateRectangle(x1,y1,x2,y2,MOVE_TOKEN,Rate,Count)`

Parameters:

| | |
|--------------------------|---|
| <code>x1,y1,x2,y2</code> | co-ordinates of the client area target. These do not have to be visible in the client area. |
| <code>MOVE_TOKEN</code> | Defines the direction of the rotation, and synchronous (script waits till done) or asynchronously (script continues operation). <code>T2B[SYNC ASYNC]</code> <code>B2T[SYNC ASYNC]</code> <code>R2L[SYNC ASYNC]</code> <code>L2R[SYNC ASYNC]</code> |
| <code>Rate</code> | The speed at which the pixels are rotated. This is in effect a delay counter. At present this varies depending on the clock speed of the PC that the program is running on. |
| <code>Count</code> | The number of rotations that will be performed. A value of 0 will cause the rotation to run until the PIXCL application is terminated. |

Remarks:

The rotation process is run as a thread or lightweight process, so other commands can be issued, if an ASYNC mode is used.

Related Commands:

None.

Run

Launches another Windows application from within an PiXCL program.

Syntax: `Run(CommandLine$)`

Parameter:

`CommandLine$` A string containing the command line (filename plus optional parameters) for the application to be executed.

Remarks:

The application you launch can be a Windows 3.1, 95 or NT application, or some other type of application (for example, MS-DOS, OS/2, or POSIX) if the appropriate subsystem is available on your PC.

When PiXCL encounters a `Run` command, it launches the program specified in `CommandLine$`, and then immediately executes the next command in the script without pausing.

In previous versions of PiXCL, you could have an PiXCL program pause after executing a `Run` command. To do so, you would use a `SetWaitMode(FOCUS)` command before `Run` and a `WaitInput(1)` command immediately after. In PiXCL 4.0 and later, this technique is obsolete. Because of the 32 bit Windows multitasking architecture, there are two ways to pause PiXCL after executing a `Run` command, using either

a) a variation of the following method...

```
Run(CommandLine$)
WinGetActive(Win$) {returns the name string}
Loop:
WinExist(Win$,Result)
WaitInput(1000)
If Result = 1 Then Goto Loop
Continue:
```

or if the second application is also a PiXCL program, by using a `WaitInput()` command, and have the second application send a `PXLResume` command to the first PiXCL application.

If `CommandLine$` does not contain a directory path, Windows searches for the executable file in the following order:

1. The current directory.
2. The Windows system directory.
3. The Windows directory.
4. The directories listed in the PATH environment variable.

If you want to change the size or location of an application window, use `WinLocate`. To hide, unhide, maximize, minimize, or restore a window, use `WinShow`. If you want the location of any window, use the `WinGetLocation` command.

Examples:

This example launches Notepad and has it automatically load the AUTOEXEC.BAT file on startup. Without pausing, the program then launches CMD.EXE.

```
Run ("NOTEPAD C:\AUTOEXEC.BAT")  
Run ("C:\WINDOWS\SYSTEM32\CMD.EXE")  
WaitInput()
```

Related Commands:

[RunExt](#), [WaitInput](#), [WinClose](#), [WinGetLocation](#), [WinLocate](#), [WinShow](#)

RunExt

This command lets you launch another application from within an PiXCL program and, at the same time, control the priority of its main process. You can also control the program's environment, current directory, window size, and window placement.

Syntax:

```
RunExt(CommandLine$,IDLE/NORMAL/HIGH,Environment$,CurrentDir$,  
x1,y1,x2,y2,WAIT/NOWAIT,WaitInit)
```

Parameters:

| | |
|----------------------|--|
| <i>CommandLine\$</i> | A string containing the command line (filename plus optional parameters) to be used to launch an application. |
| <i>IDLE</i> | Sets the priority of the application's process to level 4, the same priority level as a screen saver. |
| <i>NORMAL</i> | Sets the priority of the application's process to level 9 when the application is in the foreground and to level 7 when the application is in the background. You should use <i>NORMAL</i> for most applications. |
| <i>HIGH</i> | Sets the priority of the application's process to level 13, the same priority level as the Task Manager. You should use this setting only when absolutely necessary. |
| <i>Environment\$</i> | A string that specifies the environment to be used by the application. If the string is null (" "), PiXCL's environment is used. The environment variables within the string should be separated from one another by commas (see the first example). |
| <i>CurrentDir\$</i> | A string that specifies the current drive and directory for the application. If the string is null (" "), PiXCL's current drive and directory are used. |
| <i>x1,y1</i> | The upper-left corner of the window. |
| <i>x2,y2</i> | The lower-right corner of the window. |
| <i>WAIT</i> | Wait until the application's main process is waiting for user input with no input pending, or until the <i>WaitInit</i> interval has elapsed. (This setting has no effect on Windows 3.x apps.) |
| <i>NOWAIT</i> | Do not wait until the application's main process has no input pending. |
| <i>WaitInit</i> | If <i>WAIT</i> is specified for the previous parameter, this is the interval in milliseconds to wait. |

Remarks:

The purpose of this command is to give you more control over an application as it is launched -- for example, to specify its environment, current directory, window size, and window location. It is also intended to give you control over the priority of the application's main process. Although it's uncommon to need to control the process's priority, you may find it helpful when sending keystrokes to Windows 3.x programs using *SendKeys*. (See the *SendKeys* and *SetPriority* commands for more on this topic.)

See the *SetPriority* command for further explanation of the *IDLE*, *NORMAL*, and *HIGH* settings.

By setting *x1,y1,x2,y2* all to 0, you can have Windows determine the size of the window that is created (i.e. Windows uses default settings).

The most common reason to use the [WAIT](#) parameter with a [WaitInit](#) setting is when you want to send keystrokes to an application (using [SendKeys](#)) immediately after launching it and you want to make sure the application is ready to accept the input.

Examples:

This example launches CMD.EXE and, in the process, sets its window size to 400 by 300 pixels. It also sets the PATH and PROMPT environment variables.

```
UseCoordinates(PIXEL)
RunExt("CMD.EXE",
    NORMAL,
    "PATH=C:\WINDOWS\system32;C:\WINDOWS,PROMPT=Bob's$P$G",
    "c:\temp",
    10,10,
    400,300,
    NOWAIT,
    0)
```

Related Commands:

[SendKeys](#), [SetPriority](#), [SetSendKeysPriority](#), [Run](#)

SaveArray

PIXCL 5 command. The contents of array variables can be saved into binary files with the [SaveArray](#) command, and later restored with the [RestoreArray](#) command. The files are binary, with the the following format:

Bytes 0-3 = the number of entries in the array (current maximum 2048K);

Bytes 4-5 = the array type (1 = string, 3 = integer, 5 = float);

Bytes 6-7 = the number of arrays in the save file.

The array data follows: for integer and float arrays, each entry takes four bytes. For string arrays, each string entry is followed by two pipe characters i.e. "|". The last four bytes are a delimiter 0xfffffff, which is used only in multisave arrays. The contents of the file must match the size and type of the target array. A saved array of **n** elements can be restored to an array of **n+m** elements.

Syntax: [SaveArray\(ArrayVariable,SaveFileName\\$,Result\)](#)

Parameters:

| | |
|--------------------------------|--|
| ArrayVariable | A pre-existing array variable, in the form Name[0] , Name\${0} or Name&[0] . |
| SaveFileName\$ | The path and name of the file into which the array is to be saved. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[Array](#) [FreeArrayVar](#) [FreeArrayVarAll](#) [RestoreArray](#)

SaveBitmap, SaveBitmapChannel

Image processing operations create modified bitmaps that have to be re-displayed with a [DrawBitmap](#) or [DrawSizedBitmap](#) or [DrawZoomedBitmap](#) command. Using these commands sets the specified image as the current bitmap, at which point the [SaveBitmap](#) command can be used. This writes the current bitmap out to the disk in the selected format, either as a new image, or overwriting a pre-existing image of the same name. No check is done to see if an image is being overwritten: this is your responsibility. Use the [FileExist\(\)](#) command before the [SaveBitmap](#) command if this is an issue.

One use of [SaveBitmap](#) is automated format conversion of images. You could create a list of images and sequentially read them into memory (e.g. with [LoadBitmap](#) or [DrawBitmap](#)), then save them out in the new format.

[SaveBitmapChannel](#) is generally used to save an individual channel of a colour composite image.

Syntax: [SaveBitmap\(ImageFilename\\$,Result\)](#)

[SaveBitmapChannel\(Handle, ImageFilename\\$,Result\)](#)

Parameters:

[Handle](#)

A bitmap handle returned from [GetChannel](#) or [TuneImage](#) or [SetCurrentBitmap](#).

[ImageFilename\\$](#)

The name of the output file that the bitmap is to be written. Not all variations of readable image formats can be written. Supported formats for writing are BMP, JPEG, PCX, PPM, PSD, SGI, RAS, RLE, TIF and TGA. Writing Run Length Encoded PNG images are not supported. Photo-CD is a read-only format proprietary to Kodak. Formats are 24 bits per pixel where possible, with RGB encoding.

JPEG: 24 bits per pixel, with standard encoding.

BMP: 24 bits per pixel, RGB encoding

[Result](#)

1 if the bitmap was saved successfully, otherwise 0.

Remarks:

PIXCL creates a double linked list of bitmap records each time a [Draw\[Sized\]Bitmap](#) command is issued the first time for a specific image file. The bitmap data is stored in main memory, and a (sized) copy is written to the PIXCL memory and display contexts. This is why a copy drawn in the client area remains visible even if you issue a [FreeBitMap](#) or [FreeBitmapAll](#) command. Once a bitmap is stored in memory, subsequent calls to [Draw\[Sized\]BitMap](#) reads the bitmap data from memory, not from the disk file. Hence, if you have a file that changes on disk between display commands, you must issue a [FreeBitmap](#) command so that the new file on disk will be displayed.

[SaveBitmap](#) looks at the specified bitmap format, and saves the output image in the same number of bits per pixel, if possible. Not all readable formats are writeable. For example, you can read a Kodak Photo-CD (PCD) file, but you cannot write one. Similarly, if the input image is 24 bits per pixel, you cannot save this as an RLE file, as this supports 8 bit images only.

If the specified image format is not supported, a MessageBox will appear to inform you, and the call to [SaveBitmap](#) returns a 0.

PIXCL 5: A common programming error is that the current bitmap (loaded say as a JPEG) has to be saved to a TIF. Both of these formats support Options fields, which are incompatible. For this example you should change the name of the list image with [RenameListImage](#), that creates a new Options field, then set the desired options with [SetTIFOptions](#) command. A typical error result is that a 0 byte TIF file is created on the disk.

Consequently, when you want to save a bitmap back to disk, perhaps in an alternative format, you must select the desired stored bitmap to the current bitmap. For example:

[SaveAStoredImage:](#)

```
DrawBackground
FreeBitmapAll
ImageName1$ = SourceDir$ + "\liddnext.jpg"
ImageName2$ = SourceDir$ + "\liddxxxx.bmp"
DrawSizedBitmap(10,10,250,220,ImageName1$)
ScatterPixels(25,Res)
{DrawSizedBitmap(10,10,250,220,ImageName1$)}
SaveBitmap(ImageName2$,Res)
Goto Wait_for_Input
```

This code fragment above draws the original image, then performs a scatter pixel operation on the image in memory, without displaying. Uncommenting the second [DrawSizedBitmap](#) command will display the modified bitmap. The JPEG bitmap is saved to disk as a 24-bit BMP.

```
TestSaveAStoredImage:
FreeBitmapAll
DrawBackground
DrawSizedBitmap(10,10,250,220,ImageName1$)
DrawSizedBitmap(260,10,500,220,ImageName2$)
Goto Wait_for_Input
```

The code fragment above follows on from the first fragment. All of the stored images in memory are deleted, and the original and saved bitmaps are read in and displayed.

Related Commands:

[ConvertColorSpace](#) [SaveRectangle](#) [SaveTIFcompressMode](#)

SavedArrayInfo

PIXCL 5 command. The file in which you save arrays can be queried to extract information. This is especially useful when you need to create the necessary arrays prior to a [RestoreArray](#) operation.

Syntax: [SavedArrayInfo\(ArrayFile\\$, SaveIndex, ArrayType, Elements, NumberOfArrays\)](#)

Parameters:

| | |
|--------------------------------|---|
| ArrayFile\$ | The name of the saved array file. The array save file must exist or the command returns 0 in all other variables. |
| SaveIndex | Set to 0 to return just the NumberOfArrays in the file, otherwise 1 or greater. |
| ArrayType | The selected array type located in the file. 1 = string, 3 = integer, 5 = floating point. |
| Elements | The number of elements in the selected array located in the file. |
| NumberOfArrays | The number of arrays saved in the file. For a valid array save file this will be at least 1, otherwise 0. If SaveIndex >= the number of arrays in the file, NumberOfArrays returns 0. |

Related Commands:

[SaveArray](#) [SaveMultiArray](#)

SaveImageColorMap

PIXCL 5 command. For 8 bit images loaded into the image list, you can save the current colour map into a palette (**.PAL**) file for later use.

Syntax: `SaveImageColorMap(ListImageName$, FULL|PREVIEW,PALfile$,Result)`

Parameters:

| | |
|------------------------------|--|
| <code>ListImageName\$</code> | The name of the image loaded into your PiXCL application. This must be 8 bits per pixel, or Result returns 0. 24 bit images do not have a colour map. |
| <code>FULL PREVIEW</code> | Defines whether the image was loaded in full or preview mode. In most instances, you will load the image in FULL mode eg with DrawBitmap . |
| <code>PALfile\$</code> | The name of the file to be created. If the file exists it is overwritten. |
| <code>Result</code> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[LoadImageColorMap](#) [CreatePALfile](#) [DrawBitmap](#)

SaveMultiArray

PIXCL 5 command. The contents of several array variables can be saved into binary files with the [SaveMultiArray](#) command, and later restored with the [RestoreMultiArray](#) command. The files are binary, with the the following format:

Bytes 0-3 = the number of entries in the array (current maximum 2048K);

Bytes 4-5 = the array type (1 = string, 3 = integer, 5 = float);

Bytes 6-7 = the number of arrays in the save file.

The array data follows: for integer and float arrays, each entry takes four bytes. For string arrays, each string entry is followed by two pipe characters i.e. "|". The last four bytes are a delimiter 0xffffffff, which is used only in multisave arrays. The contents of the file must match the size and type of the target array. A saved array of **n** elements can be restored to an array of **n+m** elements.

Syntax: [SaveMultiArray\(ArrayVariable_1, ..., ArrayVariable_n, SaveFileName\\$,Result\)](#)

Parameters:

| | |
|--------------------------------|--|
| ArrayVariable | A pre-existing array variable, in the form Name[0] , Name\${0} or Name&[0] . |
| SaveFileName\$ | The path and name of the file into which the array is to be saved. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Commands:

[Array](#) [FreeArrayVar](#) [FreeArrayVarAll](#) [RestoreArray](#) [SaveArray](#) [RestoreArray](#) [RestoreMultiArray](#)

SaveRectangle, SaveRectangleToList

This command is used to capture a region of the PiXCL application client area, and save it in a bitmap to disk. For example, you can dynamically create a series of client area images, and save them out to a series of disk images, or save the region as a variety of image formats, for later playback with the [DrawBitmap](#) command.

Syntax: `SaveRectangle(x1,y1,x2,y2,ImageName$,FOREGND | BACKGND, Result)`

PIXCL 5 `SaveRectangleToList(x1,y1,x2,y2,ImageName$,FOREGND | BACKGND, Result)`

Parameters:

x1,y1,x2,y2

The PiXCL application client area region to be captured. The region specified does not have to be visible. You can use the [WinGetClientRect](#) command to get the current coordinates of the visible client area. Note that when PiXCL starts it creates a bitmap the size and pixel depth of the current screen e.g. optimally 1024x768x16-bit for a 2MB video card. This is the memory bitmap that is accessed to create the image on disk. This on-disk bitmap is saved as a 24-bit BMP file, regardless of the pixel depth of your current screen. If the *x2* value results in line length which is not a multiple of four pixel, *x2* is adjusted to the next multiple e.g. 101 pixels requested results in 104 pixels selected. If all four coordinates are set to 0, the current screen is captured.

ImageName\$

The filename to be used to save the image. BMP is the only supported save format at present. If another image format is used, the operation fails and returns 0. In **PIXCL 5**, a rectangle saved to the memory list can be saved in any supported format.

FOREGND | BACKGND

Save the region in the screen or memory bitmap. If the client area is off screen or hidden (see [WinShow](#)), **FOREGND** will grab whatever is visible in the screen i.e. a simple screen grabber function. In most cases you will want to use **BACKGND** mode.

Result

If the save operation succeeds, *Result* returns 1, otherwise it returns 0.

Remarks:

Internally, this function does the following. If the whole screen area bitmap is to be saved (e.g. coords are all 0), the PiXCL memory bitmap is used as the source. If a region is to be saved, a new bitmap region is created in memory, saved to disk, then the new region is deleted and the memory returned to the global heap.

Related Commands:

[SaveBitmap](#)

SaveStdProfileSettings

PIXCL 5 command. All new applications should keep program parameters in the Registry. The [SaveStdProfileSettings](#) command (for now) just stores the last window position in screen coordinates. If the Registry key does not exist, it is automatically created.

Syntax: [SaveStdProfileSettings\(DeveloperName\\$,AppName\\$,sx1,sy1,sx2,sy2,Result\)](#)

Parameters:

| | |
|---------------------------------|---|
| DeveloperName\$ | The name of the application developer or company name. This reads from the HKEY_CURRENT_USER tree, Software\DeveloperName\$\AppName\$\Settings\WindowPos key. |
| AppName\$ | The name of your application. |
| sx1,sy1,sx2,sy2 | The screen coordinates of the application when it was last run. These are encoded to the string written to the Registry. |
| Result | 1 if the operation was successful, otherwise 0. |

Related Command:

[LoadStdProfileSettings](#)

SaveTIFcompressMode

PIXCL supports several modes of compression when saving a TIFF bitmap with the [SaveBitmap](#) command.

Syntax: [SaveTIFcompressMode\(MODE_TOKEN\)](#)

Parameters:

| | |
|----------------------------|--|
| NONE | Save file without compression. |
| CCITT_FAX3 | Save file as a Group 3 fax. Relevant to monochrome bitmaps only. |
| CCITT_FAX4 | Save file as a Group 4 fax. Relevant to monochrome bitmaps only. |
| JPEG | Save a 24 bit image with JPEG compression. |
| PACKBITS | Save file with PackBits compression. |
| DEFLATE | Save file with the Deflate compression scheme. |

Remarks:

Selecting FAX3 or FAX4 compression mode with other than a monochrome (1 bit) image will cause a [SaveBitmap](#) or [SaveBitmapHandle](#) call to fail.

Related Commands:

[ReadTIFcompressMode](#) [SaveBitmap](#) [SaveRectangle](#)

Scrollbar

When you need to enter a numeric control variable, and instead of typing the value into an edit control, it is more logical to use a slider control called a Scrollbar, especially when working with images.

Scrollbars can be vertical or horizontal, and come in several styles. When a Scrollbar control is used, it

a) takes the mouse and keyboard focus; and

b) when you release the mouse, jumps to the label in your script and executes the commands found there.

You can have up to 8 Scrollbars visible at any one time, and you can selectively update the range values, set the slider position, get the current slider value, and delete one or more of the controls, using the related commands, [ScrollbarSetRange](#), [ScrollbarSetPosition](#), [ScrollbarGetValue](#) and [ScrollbarRemove](#), respectively.

Syntax: [Scrollbar\(x1,y1,x2,y2,VERT|HORZ,Min,Max,ScrollbarNumber, Label\)](#)

Parameters:

| | |
|---------------------------------|---|
| x1,y1,x2,y2 | The client area position of each trackbar. |
| VERT HORZ | Vertical or horizontal orientation. |
| Min, Max | Positive or negative range. |
| ScrollbarNumber | Zero if the operation failed, otherwise a number in the range 1-8 |
| Label | Jump to label for Scrollbar actions. |

Remarks:

See sample program "[controls.pxl](#)".

Related Commands:

[ScrollbarSetRange](#) , [ScrollbarSetPosition](#) , [ScrollbarGetValue](#) , [ScrollbarRemove](#)

ScrollbarGetValue

Retrieve the current scrollbar value

Syntax: [ScrollbarGetValue\(SBnumber, Value\)](#)

Parameters:

[SBnumber](#) The number in range 1-8 of the scrollbar.

[Value](#) The current value according to the range set in the [Scrollbar](#) command.

Related Commands:

[Scrollbar](#) , [ScrollbarSetRange](#) , [ScrollbarSetPosition](#) , [ScrollbarRemove](#)

ScrollbarRemove

Closes the scrollbar window.

Syntax: [ScrollbarRemove\(SBnumber\)](#)

Parameters:

[SBnumber](#) The number in range 1-8 of the [Scrollbar](#) to remove. If [SBnumber](#) is 0, all scrollbars are removed.

Related Commands:

[Scrollbar](#) [ScrollbarSetRange](#) [ScrollbarSetPosition](#) [ScrollbarGetValue](#)

ScrollbarSetPosition

Use this command to set a new position of the slider.

Syntax: [ScrollbarSetPosition\(ScrollbarNumber,Position\)](#)

Parameters:

ScrollbarNumber The number in range 1-8 of the Scrollbar.

Position New numeric position of the slider.

Related Commands:

[Scrollbar](#) , [ScrollbarSetRange](#) , [ScrollbarGetValue](#) , [ScrollbarRemove](#)

ScrollbarSetRange

Use this command to reset the range, tick mark frequency and strings of an existing [Scrollbar](#) control. This is useful when you want to change only one of a group of controls.

Syntax: [ScrollbarSetRange\(ScrollbarNumber,Min,Max\)](#)

Parameters:

[ScrollbarNumber](#) The number in range 1-8 of the Scrollbar.

[Min, Max](#) Positive or negative range

Related Commands:

[Scrollbar](#) [ScrollbarSetPosition](#) [ScrollbarGetValue](#) [ScrollbarRemove](#)

SCSIHostAdapterCount

PIXCL 5 command. Windows 95/98/ME access SCSI class devices using the Adaptec (tm) ASPI Manager **windows\system\wnaspi32.dll**. This DLL provides a standardised interface to SCSI devices. **SCSIHostAdapterCount** needs to be called once in your applications to initialize the ASPI Manager, and should ideally be run before your application exits, to ensure that device I/O is cleaned up completely.

It is possible that ASPI is not installed on your Win 95/96/NT4 (but not Win ME or 2000) PC. There are two utilities from Adaptec:

- a) **ASPICHK.exe** that checks the version and functionality of ASPI. Final version is 4.60; and
- b) **ASPI32.exe** that installs ASPI on your PC.

These are available from www.adaptec.com or from the VYSOR Support Pages www.vysor.com

Syntax: **SCSIHostAdapterCount(ASPI_ENABLE|ASPI_DISABLE,Count)**

Parameters:

ASPI_ENABLE

Loads **windows\system\wnaspi32.dll** and initializes the APSI Manager

ASPI_DISABLE

Unloads **windows\system\wnaspi32.dll**

Count

On **ASPI_ENABLE**, The number of SCSI host adapters located. This will almost always be at least 1, as CD-ROM devices are considered to be SCSI devices, even if they are actually IDE or E-IDE.

Related Commands:

[SCSIGetDeviceType](#)

SCSIGetDeviceType

PIXCL 5 command. There are different types of SCSI devices, the most common being disks and tapes, but other devices are also available.

| | |
|-------|--|
| 0 | Disk Device |
| 1 | Tape Device |
| 2 | Printer |
| 3 | Processor (uncertain what this might be) |
| 4 | Write-once read-multiple |
| 5 | CD-ROM device (includes DVD-ROM) |
| 6 | Scanner device |
| 7 | Optical memory device |
| 8 | Medium Changer device |
| 9 | Communications device |
| 10-30 | Reserved values |
| 31 | Unknown or no device type |

Syntax: `SCSIGetDeviceType(HostAdapter, DevID, LUN, DevName$, Type, Type$)`

Parameters:

HostAdapter

A value of 0 to 7. The maximum number of host adapters supported is 8.

DevID

A value of 0 to 7 for SCSI and SCSI-2 adapters, and 0 to 15 for SCSI-3 devices.

LUN

Logical Unit Number. Almost always 0, as LUNs are rarely used in modern devices.

DevName\$

The name reported by *DevID*. If no device is present at *DevID*, returns an empty string.

Type, Type\$

The device type number and decoded into a string. If no device is present at *DevID*, returns -2.

Remarks:

This command does not report the Host Adapter name, as it is not strictly considered to be a device. Use the [SCSIHostAdapterInquiry](#) command to get the desired information.

Related Commands:

[SCSIHostAdapterInquiry](#)

SCSIHostAdapterInquiry

PIXCL 5 command. Get the SCSI id# and name of a host adapter.

Syntax: `SCSIHostAdapterInquiry(HostAdapter,HostID,HA_Name$,HA_Idenifier$)`

Parameters:

HostAdapter

A value of 0 to 7. The maximum number of host adapters supported is 8.

HostID

The returned SCSI id# of the specified host adapter. This is usually 7 or 15, but may in cases be other values depending on how the hardware has been set up.

HA_Name\$

String describing the SCSI manager. If you have multiple hardware adapters from different manufacturers, these strings will be different.

HA_Idenifier\$

The name reported by the specified host adapter.

Related Commands:

[SCSIGetDeviceType](#)

SCSIReset

PIXCL 5 command. SCSI devices support a reset command that is equivalent to cycling the power on the device. For example, tape devices and scanners (and even disks) can get themselves into a problematic situation that a reset only will cure.

Syntax: *SCSIReset(HostAdapter, DevID, Result)*

Parameters:

HostAdapter

A value of 0 to 7. The maximum number of host adapters supported is 8.

DevID

A value of 0 to 7 for SCSI and SCSI-2 adapters, and 0 to 15 for SCSI-3 devices.

Result

1 if the operation completed successfully, 0 if the operation was aborted by the ASPI Manager , and -1 if the operation was aborted by the ASPI Manager because of a failure of some kind.

Related Commands:

[SCSIGetDeviceType](#)

SendCopyDataMsg

When you need to asynchronously pass text information to another application without using the clipboard, the [SendDataMsg](#) command is very useful. In PiXCL, it is provided to enable

- a) one PiXCL application to send information to another, and have the target PiXCL application start processing at a specific label; and
- b) applications written in other languages like Visual Basic, C and C++ to communicate easily with a PiXCL application.

The complementary [GetCopyDataMsg](#) command is used by a target PiXCL application to retrieve the message string.

Syntax: [SendCopyDataMsg\(WindowName\\$,Message\\$\)](#)

Parameters:

[WindowName\\$](#) The target window title. If this is NULL, or the window does not exist, the command has no effect. In addition, if the target application does not know how to process the message, or the message syntax is unrecognised, the target application SHOULD ignore it, but equally, its behavior may be unpredictable.

[Message\\$](#) The message that is to be sent to the target application. A PiXCL target application requires a specific message syntax: see the remarks below. Other applications will also generally require their own message syntax.

Remarks:

The PiXCL received message syntax must be in the form [Label_Name Message_String](#) with a space delimiter before the [MessageString](#), if it exists. The length of the [MessageString](#) can be of arbitrary length, including 0. The PiXCL target application must have a defined label of [Label_Name](#), or the message is ignored, and there should generally be [GetCopyDataMsg](#) command immediately following [Label_Name](#), unless you know that the message will be a NULL string.

See the [comms1.px1](#) and [comms2.px1](#) programs for a simple example of using the [SendCopyDataMsg](#) and [GetCopyDataMsg](#) commands.

Related Commands:

[GetCopyDataMsg](#) [PXLresumeAt](#)

SendKeys

Sends keystrokes to the named Windows 95 or NT or Windows 3.x graphical application.

Syntax:

```
SendKeys(Windowname$,Keystrokes$,PauseRespond,PauseKeystroke,Respond)
```

Parameters:

| | |
|-----------------------|---|
| <i>Windowname\$</i> | The name of the Windows 95, NT or Windows 3.x graphical application you want to send keystrokes to. Be sure to specify the full window name exactly as it appears in the Task List, paying careful attention to spacing (case doesn't matter). |
| <i>Keystrokes\$</i> | The keystrokes you want to send in the form of a string. |
| <i>PauseRespond</i> | The number of milliseconds to pause when waiting for a window to respond to <i>SendKeys</i> 's initial effort to send messages to it. Start with a setting of 500 and increase or decrease it as necessary. Decreasing is often more useful. |
| <i>PauseKeystroke</i> | The maximum number of milliseconds to pause between keystrokes. Under most circumstances, PiXCL will not pause the full amount of time but will continue on as soon as it can. Start with a setting of 500 and increase or decrease it as necessary. |
| <i>Respond</i> | An integer variable that indicates whether <i>Windowname\$</i> (or its descendants) responded to <i>SendKeys</i> 's initial effort to send messages to it. If it did respond, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

SendKeys works only with Windows 95, NT and Windows 3.x graphical applications. It does not let you send keystrokes to character-based applications, including Windows NT console, DOS, OS/2, and POSIX applications.

Don't expect too much from this command, because sending keystrokes to other applications in a multitasking environment like Windows 95 and NT doesn't always work. For example, you may send a pattern of keystrokes several times in a row without incident, but then the operation will fail for no apparent reason. This is the nature of the operating system. At any given time, Windows (or an active application) may be using the system's resources to the extent that the target application window cannot respond to *SendKeys*'s effort to send messages to it. When this happens, the *Respond* variable is set to 1, indicating failure. (The same thing may happen when the keystrokes you're sending cause the target's parent window to change, for example, a new dialog box to appear. Again, *SendKeys* must determine whether the new target window is responding to messages being sent to it.)

You'll have the best luck with *SendKeys* if you send short sets of keystrokes and break up your keystrokes into more than one command.

Placing a *SetSendKeysPriority(NORMAL)* immediately before the *SendKeys* command is often useful.

Here are the ways to represent the CTRL, ALT, SHIFT, and ENTER keys within the *Keystroke\$* argument:

SHIFT +
ALT %
CTRL ^
ENTER ~

The table below shows how to represent certain other keys.

| Keystroke | Representation |
|------------|---------------------|
| ===== | ===== |
| BACKSPACE | {BACKSPACE} or {BS} |
| BREAK | {BREAK} |
| CAPSLOCK | {CAPSLOCK} |
| CLEAR | {CLEAR} |
| DEL | {DEL} or {DELETE} |
| DOWNARROW | {DOWN} or {D} |
| END | {END} |
| ENTER | {ENTER} or ~ |
| ESC | {ESC} or {ESCAPE} |
| F1 to F24 | {F1} to {F24} |
| HELP | {HELP} |
| HOME | {HOME} |
| INS | {INSERT} |
| LEFTARROW | {LEFT} or {L} |
| NUMLOCK | {NUMLOCK} |
| PGDN | {PGDN} |
| PGUP | {PGUP} |
| PRINTSCRN | {PRTSC} |
| RIGHTARROW | {RIGHT} or {R} |
| SCROLLLOCK | {SCROLLLOCK} |
| SPACE | {SPACE} or {SP} |
| TAB | {TAB} |
| UPARROW | {UP} or {U} |
| % | {%} |
| + | {+} |
| ^ | {^} |
| { | {{} |
| } | {}} |
| ~ | {~} |

Table: **Keystroke Representations for [SendKeys](#)**

Be sure not to include any spaces in your [Keystrokes\\$](#) argument. If you want to send space characters to an application, use {SPACE} instead.

You can express duplicate keystrokes by placing the keystroke within curly braces and using a repetition count. For example, {t 3} is the same as **ttt** and {SPACE 2} is the same as {SPACE}{SPACE}.

If your keystrokes cause a new control or window to become active, don't use a repetition count with them--for example, rather than {TAB 3}, use {TAB}{TAB}{TAB}. (Using a repetition count in such a situation will cause PiXCL to misinterpret the context, and your keystrokes will not work.)

[PauseRespond](#) is the maximum amount of time PiXCL waits for a window to initially respond to keystrokes. (PiXCL also uses this value before sending the first keystroke and each time a keystroke causes the target window to change--for example, a new dialog box or application window to appear.) A setting of 500 is usually best, but you may need to increase it if you find that PiXCL isn't sending keystrokes to the proper window.

Likewise, the [PauseKeystroke](#) argument controls the maximum amount of time PiXCL waits between keystrokes. Under most

circumstances, PiXCL doesn't need to wait the full time, because it can detect that the target application has received a keystroke and is ready for another. But sometimes a keystroke merely sets a setting internal to the target program and PiXCL has no way of detecting whether the keystroke has been processed. You can set [PauseKeystroke](#) to a value of 500, and then increase or decrease it based on [SendKeys](#)'s success rate sending keys. As a general rule, you should try increasing [PauseKeystroke](#) only if the keystrokes you're sending are being skipped, appear out of sequence, or don't have their desired effect as they flow to [Windowname\\$](#).

If you're having trouble sending keystrokes to a Windows 3.x application, see the [SetPriority](#) and [SetSendKeysPriority](#) commands for some possible help.

Example:

The following example sends some simple keystrokes to Notepad.

```
Program$ = "Notepad - (Untitled)"
Keys$ = "Simple keystrokes being sent to Notepad~Line 2(HOME)"
SendKeys(Program$,Keys$,
    500,          {Wait for .5 secs for window to respond}
    500,          {Wait for max of .5 secs between keystrokes}
    Respond)
If Respond = 0 Then Beep
```

Related Commands:

[RunExt](#), [SetPriority](#), [SetSendKeysPriority](#), [Run](#)

Set

This command lets you assign values to integer or string variables. In the case of integer variables, you can also perform simple mathematical calculations using integers. When assigning string variables, you can perform concatenation—that is, joining one or more strings to the end of another and storing the result.

Syntax:

```
Set Variable = -2147483647 to 2147483648
```

```
Set Variable = <math_expression>
```

```
Set Variable$ = "String"
```

```
Set Variable$ = "String1" + "String2" + Variable2$
```

or

```
Set Variable$ = <string_expression>
```

Although the **Set** portion of the command is required in previous versions of PiXCL, it is optional in PiXCL 4.0 and later. For example, the following two forms of syntax are identical:

```
Variable$ = "String"
```

```
Set Variable$ = "String"
```

Parameters:

Variable A valid integer variable name (see "Variables" earlier).

<math_expression> A mathematical expression of the form

```
Integer1 math_operator Integer2
```

where *Integer1* and *Integer2* are integers (or integer variables) and *math_operator* is one of the mathematical operators. For example, the following are all valid mathematical expressions:

```
Counter + 3  
X2 / 18  
10 * Box_size
```

Variable\$ A valid string variable name (a trailing \$ is required).

<string_expression> A string expression of the form

```
"String1" + "String2" [+ String3 ... + Stringn]
```

where *"String1"* and *"String2"* represent strings enclosed in double quotation marks (or string variable names) and + is the concatenation symbol. For example, here are some valid string expressions:

```
"Nonprofit " + "Corporation"  
Yearend$ + "Results"  
File1$ + File2$
```

| <u>Operator</u> | <u>Meaning</u> |
|-----------------|----------------|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |

Table: Mathematical Operators

Remarks:

In PiXCL 4.0 and later, integer variables can be in the range -2147483647 to 2147483648.

When assigning integer variables, the results of mathematical expressions are rounded down and stored as integers. For example, the command `Set Green=2/3` assigns a value of zero to the variable Green. Likewise, the command `Set Green=3/2` assigns a value of 1 to Green.

If you want to perform more than one mathematical operation at a time, the only way to do so is to break them up into separate operations. For example, the following command is invalid:

```
y = 3 * x + 4 {Invalid}
```

But you can accomplish the same thing using these two commands:

```
y = 3 * x
y = y + 4
```

The modulus operator can be used to perform an operation every nth iteration of a loop.

```
Modulus = 21 % 10          returns a value of 1 to Modulus
```

You can perform string concatenation with any number of strings and string variables. The command can extend over multiple lines as well. For example, to put a carriage return into a string that is to be displayed with the `DrawTextExt` command, you would use code like

```
Chr(13,cr$)
A$ = "One "
B$ = "Two "
C$ = "Three "
Six$ = "Six"
Test$ = A$ + B$ + C$ + "Four " + cr$ +
        "Five " + Six$
DrawTextExt(20,50,300,200,Test$,LEFT)
```

Examples:

To check an operation every 20 iterations to see if an exit has been called, use code like

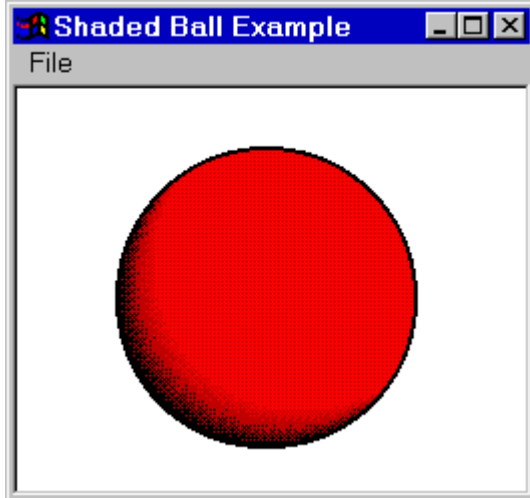
```
Index = 0
WinGetClientRect("",cx1,cy1,cx2,cy2)
SetMouse(cx1,cy1,cx2,cy2, StopDraw,X,Y)
Random(255,R) Random(255,G) Random(255,B)
Loop:
```



```

UseBrush(SOLID, R, G, B )
DrawRectangle(x1,y1,x2,y2)
Index = Index + 1
J = Index % 20
If J = 0 Then WaitInput(1) {check for input}
Goto Loop
StopDraw:
SetMouse()
Goto Wait_for_Input

```



The following program draws the 3-D ball in red, as shown at left. By modifying the [Set](#) commands at the beginning of the program, you can change the size of the ball and its position on the screen.

A 3-Dimensional Shaded ball.

```

{Initialize variables}
UseCoordinates(PIXEL)
UsePen(NULL,0,0,0,0)      {Use NULL pen for shading}
Set Red=0
Set Blue=0
Set Green=0
Set x1=50                  {Starting x position}
Set Final_x1=x1
Set y1=180                 {Starting y position}
Set Final_y1=y1
Set Ball_size=150         {Ball size}
Set Count=1
Next_shade:
If Count>10 Then Goto Flood
Set x2=x1+Ball_size
Set y2=y1-Ball_size
UseBrush(SOLID,Red,Green,Blue)
DrawEllipse(x1,y1,x2,y2)
Set x1=x1+2
Set y1=y1-2
Set Red=Red+25
Set Count=Count+1
Goto Next_shade

Flood:
UsePen(SOLID,2,0,0,0)

```

```

UseBrush(NULL,0,0,0)
Set x2=Final_x1+Ball_size
Set y2=Final_y1-Ball_size
DrawEllipse(Final_x1,Final_y1,x2,y2)
UseBrush(SOLID,255,255,255) {White brush}
DrawFlood(1,1,0,0,0) {Flood with white}
WaitInput()

```

This next example shows how to assign string variables and perform string concatenation using [Set](#). In particular, it illustrates how to use the [Chr](#) function to handle the thorny issue of placing double-quotation marks within a string.

```

{Get first name using TextBox}
Top: Text$ = "What is your name?"
Caption$ = "Enter name"
TextBox(Text$,Caption$,Input$,Button)
{If Cancel mashed, then quit}
If Button = 2 Then End
{Build MessageBox prompt with double quotes around name,
for example, 'Is "George" correct?'}
Chr(34,Quote$)
Prompt$ = "Is " + Quote$ + Input$ + Quote$ + " correct?"
{Put up message box}
MessageBox(YESNO,1,QUESTION,Prompt$,"Verify",Button)
{If No mashed, then loop to get name again}
If Button = 2 Then Goto Top:

```

Related Commands:

[DrawNumber](#), [DrawText](#), [DrawTextExt](#)

SetBitmapResolution

An image loaded into or created in the PIXCL image list can have a print resolution value in dots per inch set. The default setting is 0, that is, no resolution information is used. This is only relevant when the image is saved to a format that supports a resolution value in the file header (e.g. BMP and TIF).

Syntax: [SetBitmapResolution\(*Xdpi*, *Ydpi*\)](#)

Parameters:

[Xdpi](#), [Ydpi](#) The dots per inch in the X and Y axes.

Related Commands:

[GetBitmapResolution](#)

SetBMWMouse, SetBMWRightMouse

PIXCL provides up to eight bitmap windows with vertical and horizontal scroll bars, and a zoom capability. It is often handy to be able to select a specific pixel in a bitmap window and get the pixel coordinates. The [SetBMWMouse](#) and [SetBMWRightMouse](#) and [SetBMWMidMouse](#) commands work similarly to the [SetMouse](#) commands, and support multiple definitions in a valid command.

Syntax: [SetBMWMouse\(WindowID,Label,x,y, ...\)](#)
[SetBMWMidMouse\(WindowID,Label,x,y, ...\)](#)
[SetBMWRightMouse\(WindowID,Label,x,y, ...\)](#)

Parameters:

| | |
|--------------------------|--|
| WindowID | The bitmap window ID returned from a DrawBitmapWindow command. |
| Label | The jump-to label when the left, middle or right mouse event occurs. |
| x, y | The BITMAP pixel coordinates, based on (0,0), according to the zoom factor and roam position of the bitmap window. |

Remarks:

As with the [SetMouse](#) commands, a [SetBMW\[Right\]Mouse\(\)](#) command deletes the settings and disables the mouse functions in the bitmap window. Note that deleting the settings disables mouse actions on ALL bitmap windows. Hence, if you want to disable mouse actions on one bitmap window, you have to issue a new command.

It is recommended that when you want to get zoomed pixel coordinates, that you select whole number zoom values, as fractional zoom values can occasionally cause the reported coordinates be off by 1, due to integer rounding during the coordinate calculation process. The top-left corner of a zoomed pixel in a bitmap window should be considered the origin point of the image.

Related Commands:

[SetMouse](#) , [CloseBitmapWindow](#) , [DrawBitmapWindow](#) , [FlashBMWindow](#) , [ZoomBitmapWindow](#)

SetBreakpoint

PIXCL 5 command. Pauses the interpreter at the current point in the script and updates the debug dialog with the current the integer, float, string, array and bitmap records.

Syntax: [SetBreakpoint](#)

Parameters:

None

Related Command:

[Breakpoints](#)

SetColorPalette

Controls whether PiXCL draws a bitmap using the bitmap's own color palette or generates an evenly distributed color palette and uses it instead.

Syntax: [SetColorPalette\(BITMAP/GENERATE\)](#)

Parameters:

BITMAP

When drawing a bitmap, reads the color palette from the bitmap's file. This is the default.

GENERATE

When drawing a bitmap, uses an evenly distributed color palette and disregards the palette in the bitmap's file.

Remarks:

If you draw one 256-color bitmap after another in the same PiXCL window, the first bitmap will likely change color as the second bitmap's color palette is realized; this occurs instantly, just before the second bitmap is appears on the screen (the second bitmap appears normally). A 256 color display has 20 colors preassigned to the so-called "static colormap", and all additional colors required must use the remaining 236 color palette entries.

The [SetColorPalette](#) command was introduced to try to mitigate this problem. By using [SetColorPalette\(GENERATE\)](#), you can have PiXCL generate an evenly distributed color palette that is used by both bitmaps. Neither bitmap will appear optimal, but one bitmap's colors will not override the other's.

You must have at least a 256-color video driver installed to see the effect of this command. If you have a 2MB or greater video card, your computer will be able to display images using 65,536 colors or more. This produces a much better display, because Windows now has much larger static colormap, typically 4096. This means that for imaging applications, the bitmap palette issue becomes much less of a problem.

Example:

The following program draws one 256-color bitmap starting at the point (5,5) and then another starting at (130,5). As the second bitmap is drawn and its color palette realized into the device context, the first bitmap instantly changes colors.

```
DrawBitmap(5,5,"color.bmp")
DrawBitmap(130,5,"amber.bmp")
WaitInput()
```

To have both bitmaps use an evenly distributed color palette, you can add the [SetColorPalette](#) command as follows:

```
SetColorPalette(GENERATE)
DrawBitmap(5,5,"color.bmp")
DrawBitmap(130,5,"amber.bmp")
WaitInput()
```

Related Commands:

[DrawBitmap](#), [DrawSizedBitmap](#)

SetCommPort

This command requires a string that specifies device-control information. The string must have the same form as the mode command's command-line arguments. For example, the following string specifies a baud rate of 1200, no parity, 8 data bits, and 1 stop bit:

```
baud=1200 parity=N data=8 stop=1
```

The device name is ignored if it is included in the string, but it must specify a valid device, as follows:

```
COM1: baud=1200 parity=N data=8 stop=1
```

This command string can also be written with comma delimiters as
1200,N,8,1

For further information on **mode** command syntax, refer to the end-user documentation for your operating system.

PIXCL 4 Syntax: `SetCommPort(COMx,Mode$,XON|XOFF, Result)`

PIXCL 5 Syntax: `SetCommPort(COMx,Mode$,XON|XOFF, CR|LF|CRLF|ESC|NULL, Result)`

Parameters:

| | |
|-----------------------|--|
| <code>COMx</code> | Port, where x = 1 – 4 for standard PCs. PIXCL 5: If you have a multiple COM port board installed, ports 5-13 are supported. |
| <code>Mode\$</code> | Port state parameters, according to the MS-DOS mode command, e.g. "baud=9600 parity=N data=8 stop=2" |
| <code>XON XOFF</code> | Enables or disables the data stream control mode for both input and output. |
| <code>CR</code> | Received messages are terminated by a CR character. |
| <code>LF</code> | Received messages are terminated by a LF character. |
| <code>CRLF</code> | Received messages are terminated by a CRLF character pair. This is the most commonly used setting. |
| <code>ESC</code> | Received messages are terminated by an ESC character. |
| <code>NULL</code> | Received messages are terminated by a NULL character. |
| <code>Result</code> | 1 if the command was successful, otherwise 0. |

Remarks:

You can set the COMx port parameters from the control panel for system wide operation. The settings made within PiXCL only apply while your PiXCL application is running.

Related Commands:

[ClearCommPort](#) [EscCommFunction](#) [GetCommPort](#) [ReadCommPort](#) [WaitCommEvent](#) [WriteCommPort](#)

SetComputerName

You can set the name of the current computer if you have appropriate authority. This must be a string not longer than 15 characters, in the standard character set including letters, numbers, and the following symbols:

! @ # \$ % ^ & ' (. - _ { } ~

If you use any other characters, in Windows 95 the out-of-set characters will be changed to standard characters, while in Windows NT the operation will fail and the current name will not be changed.

Any name change will only take effect the next time you restart the computer.

Syntax: `SetComputerName(NewName$,Result)`

Parameters:

NewName\$ The new name for the computer. If this is a NULL string, the command is ignored.
Result 1 if the operation is successful, otherwise 0.

Remarks:

Issuing a `GetComputerName` after this command will not return the new name, and will usually return a null string.

Related Commands:

[GetComputerName](#)

SetConsoleTitle

PIXCL 5 command. You can programmatically change the title of an existing console window with this command.

Syntax: [SetConsoleTitle\(*Title*\\$\)](#)

Parameter:

[Title](#)\$ The new title of the console window.

Related Commands:

[ShowConsole](#) [FreeConsole](#)

SetCtrlMouse, SetShftMouse

This commands are like the [SetMouse](#) command, except that they work with control left-mouse and shift left-mouse clicks. All mouse commands use the same syntax.

Syntax:

[SetShftMouse\(\)](#)

[SetCtrlMouse\(\)](#)

or

```
SetCtrlMouse\(Region1\_x1,Region1\_y1,Region1\_x2,Region1\_y2,Label,x,y,  
Region2\_x1,Region2\_y1,Region2\_x2,Region2\_y2,Label,x,y,  
.  
.  
Regionn\_x1,Regionn\_y1,Regionn\_x2,Regionn\_y2,Label,x,y\)
```

Remarks:

See the [SetMouse](#) command for an explanation of [SetCtrlMouse](#)'s arguments and behavior.

If a [SetMouse](#) command is already in effect and the user double-clicks the left mouse button, PiXCL will respond to the [SetMouse](#) command first before responding to the [SetDbIMouse](#) command.

Related Commands:

[SetMouse](#) [SetCtrlMouse](#) [SetDbIMouse](#) [SetRightMouse](#) [SetShftRightMouse](#)

SetCurrentBitmap

Image processing commands require that the current bitmap be selected. This can be done with a [DrawBitmap](#) or [LoadBitmap](#) command, and also with the [SetCurrentBitmap](#) command.

Syntax: [SetCurrentBitmap](#)(*ImageName*,\$,FULL | PREVIEW,*Handle*)

Parameters:

| | |
|--|---|
| ImageName \$ | The desired current bitmap loaded in the image list. |
| FULL PREVIEW | The display mode of the loaded image. |
| Handle | If the image is not loaded, <i>Handle</i> returns 0, otherwise it returns the image handle. This is the same handle that TuneImage returns. |

Remarks:

[SetCurrentBitmap](#) differs from [LoadBitmap](#) in that if the image is not loaded, it will not load it automatically.

Related Commands:

[LoadBitmap](#) [TuneImage](#)

SetMidMouse, SetCtrlMidMouse, SetDbIMidMouse, SetShftMidMouse

These commands are like the [SetMouse](#) command, except that they work with control mid-mouse, shift and double midmouse clicks. All mouse commands use the same syntax. You must have a Microsoft Intellimouse™ or equivalent installed on your system for these commands to function.

Syntax:

[SetMidMouse\(\)](#)
[SetShftMidMouse\(\)](#)
[SetCtrlMidMouse\(\)](#)
[SetDbIMidMouse\(\)](#)

or

[SetMidMouse\(Region1_x1,Region1_y1,Region1_x2,Region1_y2,Label,x,y,
Region2_x1,Region2_y1,Region2_x2,Region2_y2,Label,x,y,
.
.
Regionn_x1,Regionn_y1,Regionn_x2,Regionn_y2,Label,x,y\)](#)

Remarks:

See the [SetMouse](#) command for an explanation of [SetMidMouse](#)'s arguments and behavior.

If a [SetMidMouse](#) command is already in effect and the user double-clicks the mid mouse button, PiXCL will respond to the [SetMidMouse](#) command first before responding to the [SetDbIMidMouse](#) command.

Related Commands:

[SetMouse](#) [SetCtrlMouse](#) [SetDbIMouse](#) [SetRightMouse](#) [SetShftRightMouse](#)

SetDbIMouse

This command is like the [SetMouse](#) command, except that it works with double left-mouse clicks.

Syntax:

[SetDbIMouse\(\)](#)

or

[SetDbIMouse\(Region1_x1,Region1_y1,Region1_x2,Region1_y2,Label,x,y,
Region2_x1,Region2_y1,Region2_x2,Region2_y2,Label,x,y,
.
.
Regionn_x1,Regionn_y1,Regionn_x2,Regionn_y2,Label,x,y\)](#)

Remarks:

See the [SetMouse](#) command for an explanation of [SetDbIMouse](#)'s arguments and behavior.

If a [SetMouse](#) command is already in effect in the same area, and you double-click the left mouse button, PiXCL will respond to the [SetMouse](#) command first before responding to the [SetDbIMouse](#) command.

Related Commands:

[SetMouse](#), [SetCtrlMouse](#), [SetRightMouse](#), [SetShftRightMouse](#)

SetDrawBitmap

An image that has been loaded into the PiXCL and geoPiXCL image lists can be drawn to directly with the various Draw commands. This command replaces the background memory bitmap with the image list bitmap. Drawing is done in the background image, then that image is redrawn in the foreground.

Syntax: `SetDrawBitmap(Image,$,PREVIEW|FULL,Result)`

Parameters:

| | |
|-----------------|---|
| <i>Image</i> \$ | The name of a loaded image in the list. If the image is not loaded, <i>Result</i> returns 0, and nothing happens, which means that you should nearly always check the return value. If <i>Image</i> \$ is set to NULL, the default client area drawing surface is restored. |
| PREVIEW, FULL | Same as the <code>LoadBitmap</code> command, and defines whether a full or preview sized bitmap is to set as the drawing surface. |
| <i>Result</i> | If the function succeeds, <i>Result</i> is set to 1, otherwise 0. |

Remarks:

You should issue a `SaveBitmap` to save the newly written bitmap data. The function works with 8 and 24 bit images.

Example:

OpeningFile:

```
WaitInput(1)
DrawBitmap(20,40,Image1$)
SetDrawMode(BACKGND)
SetDrawBitmap(Image1$,FULL,Res)
UseFont("Arial",13,27,NOBOLD,NOITALIC,NOUNDERLINE,255,255,0)
UsePen(SOLID,2,255,255,0)
DrawText(30,50,"This is text written to a loaded bitmap")
DrawLine(20,20,200,200)
UsePen(SOLID,4,255,0,0)
UseBrush(CROSS,0,255,255)
DrawEllipse(20,30,80,60)
SetDrawBitmap("",FULL,Res)
SetDrawMode(BOTH)
DrawBitmap(20,40,Image1$)
Goto Wait_for_Input
```

Related Commands:

[LoadBitmap](#) [SetDrawMode](#) [DrawVECcommands](#)

SetDrawMode

PIXCL keeps a foreground and background image of what you see in the application client area. [SetDrawMode](#) provides the means to write only in the foreground or background, or both, and is generally used in simple animation programs.

For example, you can use the foreground as the main display image, and manipulate animated images, while preparing the background with a different background color or bitmap image. Only when the new image is ready is it copied to the foreground.

Syntax: [SetDrawMode\(DISABLE | FOREGND | BACKGND | BOTH\)](#)

Parameters:

| | |
|-------------------------|--|
| DISABLE | Stops any Draw[Tr][Sized][Zoomed]Bitmap command writing in either memory image. This token is not commonly used. |
| FOREGND | Draws in the foreground image (i.e. the PIXCL client area) only. |
| BACKGND | Draws in the background image only. |
| BOTH | Draws in the foreground and background images. |

Notes:

It is your responsibility to change the draw mode of you want other than the default of [BOTH](#).

Related Commands:

[DrawBackgroundRegion](#) , [SetROPcode](#) , [SetDrawMouse](#)

SetDrawMouse

The [SetDrawMouse](#) enables you to draw freehand in the PiXCL application client area with the mouse while the left button is down. The currently selected pen is used, or if a [UsePen](#) command has not been issued, the default black pen is used. You can also draw in the foreground, background, or both.

The [SetDrawMouse](#) function could be used, for example, where you need to select or indicate part of the screen display. If the left button is released, the drawing action stops and the mouse moves in the normal way without drawing.

Syntax: [SetDrawMouse](#)([FOREGND](#)|[BACKGND](#)|[BOTH](#)|[DISABLE](#))

Parameters:

| | |
|-------------------------|---|
| FOREGND | The line is drawn in the foreground only (i.e. the screen context). |
| BACKGND | The line is drawn in the background only (i.e. the memory context). The line will not be visible until the screen is refreshed, either by resizing the display or issuing the ReDraw command. |
| BOTH | The line is drawn in the foreground and background. |
| DISABLE | SetDrawMouse operations are disabled. |

Remarks:

When [SetDrawMouse](#) is enabled, the cursor changes to a pen like shape. When [SetDrawMouse](#) is disabled, the cursor is reset to the standard application default.

[SetDrawMouse](#) uses the left button to initiate the draw operation. If a [SetMouse](#) command (which also uses the left button) is in effect, the PiXCL code will be activated as well, but you can still draw with the mouse. You may find it helpful to either disable [SetMouse\(\)](#) while a [SetDrawMouse](#) is in effect, or use another of the mouse commands.

Example:

Here are a number of functions that turn on, process and turn off the mouse draw option. They could be included in a [SetMenu](#) or toolbar. Note how the mouse draws in the foreground, and if the right mouse is pressed, the background image (which is unchanged) is copied to the foreground i.e. erasing the lines previously drawn.

```
Mouse_DrawOn:
    DrawBackGround
    WinGetClientRect("", cx1, cy1, cx2, cy2)
    SetRightMouse(cx1, cy1, cx2, cy2, ReDrawMemoryDC, n, n)
    SetCtrlMouse(cx1, cy1, cx2, cy2, DrawBlock, X, Y)
    SetMouse()
    SetDrawMouse(FOREGND)
    Goto Wait_for_Input
```

```
Mouse_DrawOff:
    SetDrawMouse(DISABLE)
    SetRightMouse()
    Goto Wait_for_Input
```

```
ReDrawMemoryDC:
    ReDraw
```



```
Goto Wait_for_Input
```

```
DrawBlock:
```

```
X1 = X - 15    X2 = X + 15
```

```
Y1 = Y - 15    Y2 = Y + 15
```

```
DrawRectangle (X1, Y1, X2, Y2)
```

```
Goto Wait_for_Input
```

Related Commands:

[SetMouse](#), [SetCtrlMouse](#), [SetRightMouse](#), [SetShftRightMouse](#),

[UsePen](#)

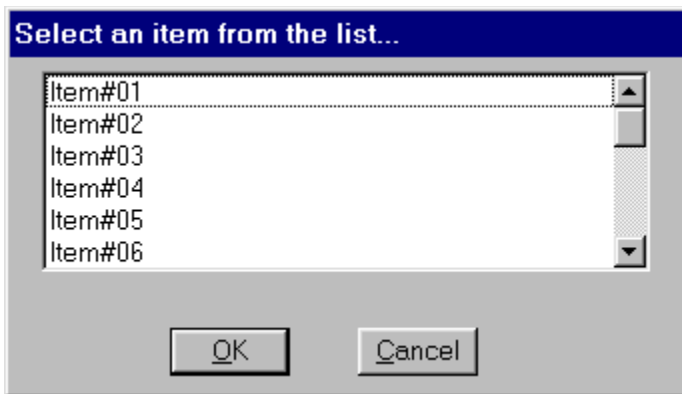
SetEditControl

This command lets you create any number of edit control windows in the PiXCL client area, for text or numeric string entry. Text entry can also be done under program control with the [SetKeyboard](#) and other commands (see the keyboard.pxl example) but is very tedious to code.

Windows manages the allocation of memory for the edit strings before they are entered into PiXCL string variables. While a string can be about 32 KB long if required, the aim of edit windows is to enter small strings such as names, numbers or program parameters.

For example, you can read an INI file or registry entries with [FileRead_INI](#) or one of the [RDB](#) commands, display the various string values in edit windows, make any desirable changes, then write the INI or registry entry back with [FileWrite_INI](#).

You can also use [SetEditControl](#) to create a form page with a variety of entries for information. For example, display an image (using [DrawSizedBitMap](#)), and create a set of edit controls to describe the image parameters, then write these parameters out to an ASCII file with [FileWrite_ASCII](#).



The above image shows the available button styles enclosed in a group box, with embedded edit controls.

Syntax:

[SetEditControl\(\)](#) to clear all edit controls

or

[SetEditControl\(x1,y1,x2,y2,STRING|NUMBER|NUMBERUD|PASSWORD,
MaxLimit, MinLimit, Text\\$,...\)](#)

Parameters:

| | |
|-----------------------------------|---|
| x1,y1,x2,y2 | Co-ordinates of the edit control. |
| STRING | Control edit text is a single line. |
| NUMBER | Control edit text is numbers only. +/- signs are not accepted from the keyboard, but can be copied from the Clipboard. |
| NUMBERUD | Control edit text is numbers only, plus an attached up-down control. +/- signs are not accepted from the keyboard, but can be copied from the Clipboard. See the example below. |
| PASSWORD | Control edit text is secure, displays * for all characters. |
| MaxLimit,MinLimit | For NUMBER edit controls only, sets the limits of integer input. Negative numbers are allowed. Values are ignored in STRING and PASSWORD modes. |
| Text\$ | Input text string. If the variable is not already defined, it is created. If it used as the default, then modified, the variable will include the changes. |

Remarks:

[SetEditControl](#) is generally used with [Button\(\)](#) commands to indicate that the edit string entry is complete. You could also set a mouse active region, or use a specific key stroke as the terminating event. Windows user interface style conventions generally suggest using a push-button or perhaps a Radio button.

It is strongly recommended that you use the [SetEditControl\(...\)](#) before the related [Button\(...\)](#) command in your script, otherwise you may get unpredictable visual results, such as pre-existing edit controls which should be redrawn, not being redrawn, or button text appearing in the system font even if a user font has been selected. This is not a bug in PiXCL, rather it is the way Windows processes commands in its message queues, and usually only appears with pushbuttons, radio-buttons and checkboxes created with [Button\(\)](#). If using [SetEditControl\(...\)](#) before the related [Button\(...\)](#) does not clear the problem, ensure that you have the latest video driver. If the problem persists, please contact VYSOR Technical Support for more information.

Edit text is displayed in the current font, either the system font, or the font selected with a [UseFont](#) command. Text is always monochrome

If you use the [NUMBERUD](#) token to create the up-down control, click the upper button to increment and the lower button to decrement the edit control value. If the variable has not been previously defined, or is not a numeric string, the first click will set it to zero. Negative numbers are possible. If you click and hold a button down (i.e. generate a stream of mouse events), the edit string will be incremented or decremented until you release the button. The resulting value is available in the variable name.

Once you have the numeric string, use the [Val](#) command to convert it to a numeric variable, and the [Negate](#) commands to take the arithmetic negative.

Example:

This code fragment draws the dialog controls shown in the image above. Some of the controls have been activated or edited once they were created.

`New_Buttons:`

```
DrawBackGround
UseFont("MS Sans Serif",0,19,
        NOBOLD,NOITALIC,NOUNDERLINE,255,0,0)
SetEditControl(20,20,125,45,STRING,0,0,Edit1$,
               20,50,125,75,NUMBERUD,0,255,Edit2$,
               20,80,125,105,PASSWORD,0,0,Edit3$)
Button(320,35,380,75,PUSH,"&OK",New_Buttons_End,
       320,85,380,125,PUSH,"&Cancel",New_Buttons_End,
       180,15,260,40,AUTORADIO,"Invert",New_Buttons_Wait,
       180,50,260,75,AUTORADIO,"Normal",New_Buttons_Wait,
       180,90,260,115,AUTOCHECK,"Check",New_Buttons_Wait,
       10,2,400,140,GROUP,"Value Selections",New_Buttons_End)
```

`New_Buttons_Wait:`

```
WaitInput()
```

Remarks:

See also the [DialogBox](#) command to produce dialog boxes with edit controls.

Related Commands:

[Button](#), [UseFont](#), [SetMouse](#), [SetKeyboard](#), [Val](#), [Negate](#)

SetEnvVariable

Use this command to set the current value of an environment variable.

Syntax: [SetEnvVariable\(Variable\\$, Value\\$, Result\)](#)

Parameters:

[Variable\\$](#) The name of the environment variable.

[Value\\$](#) The new value of [Variable\\$](#). If this is a null string, [Variable\\$](#) is deleted from the environment string list.

[Result](#) 1 if the variable was set successfully, otherwise 0.

Related Commands:

[GetEnvString](#) [GetEnvVariable](#)

SetFontEscapement

Font escapement is the angle with which character strings are written. In PiXCL, the default value is 0, which produces horizontal text. You can set this escapement value in increments of 0.1 degrees with the [SetFontEscapement](#) command. This value remains valid until changed by another [SetFontEscapement](#) command.

Syntax: [SetFontEscapement\(AngleX10\)](#)

Parameters:

[AngleX10](#)

The angle that you want text to appear, times 10. Hence, a desired angle of 15.3 degrees requires [AngleX10](#) to be 153. Positive numbers are clockwise, negative are anti-clockwise.

Remarks:

Not all fonts support the escapement setting. **MS Serif** and **MS Sans Serif** are Adobe™ fonts, and any escapement setting is ignored. In general, if the font you select is a TrueType™ font, the escapement angle can be set.

The escapement value is set in a global variable within PiXCL, and affects all subsequent [UseFont](#) commands only. This means that just setting the escapement value without a following [UseFont](#) will not have any effect.

Related Commands:

[AddFont](#) [ChooseFont](#) [GetTextSpacing](#) [DrawText](#) [DrawShadowText](#) [DrawTextExt](#) [DrawShadowTextExt](#) [RemoveFont](#)
[SetTextSpacing](#) [UseFont](#) [UseFontExt](#)

SetJPGOptions

PIXCL 5 command. When you load a JPEG image from disk, the options data in the file, if any, are also loaded and stored with the image in the PiXCL image list. These fields can be set or updated as well. If the disk image type is not JPEG, the command has no effect, and all values return null or 0. You can get this command to work if you first set change the name of the loaded image in the list, using [RenameListImage](#).

Syntax: [SetJPGOptions](#)(*Filename*\$, *Comments*\$, *ResType*, *XRes*, *YRes*)

Parameters:

[Filename](#)\$ The name of a loaded file in the image list.

Note all the following MUST be variables, or you will get a syntax error.

[Comments](#)\$ The contents of the field. This may be an empty string.

[ResType](#) The resolution indicator. 0 = no unit of measurement specified, 1 = dots per inch, 2 = pixels per meter.

[XRes](#), [YRes](#) The X and Y axis resolution.

Related Commands:

[GetJPGOptions](#) [GetPNGOptions](#) [GetTIFFOptions](#) [SetPNGOptions](#) [SetTIFFOptions](#) [RenameListImage](#)

SetKeyboard

This command lets you get keyboard input from the user. When the program is pausing for input and the user presses a specified key, the program branches to the label associated with that key.

Syntax:

`SetKeyboard()`

or

```
SetKeyboard("a",Label,  
            "^a",Label,  
            vkey,Label)
```

Parameters:

- "a"** Any white key on the keyboard (except function keys and certain keys on the numeric keypad). For example, "a" represents lowercase *a* and "R" represents uppercase *R*.
- "^a"** Any white key on the keyboard (except function keys and certain keys on the numeric keypad) in combination with CTRL. For example, "^b" represents CTRL+b and "^V" represents CTRL+V.
- vkey** A virtual key number taken from the Table below. For example, the virtual key number for the F1 function key is 112. Using a virtual key number is the only way to test for certain keys, including function keys and several keys on the numeric keypad.
- Label** A label you want PiXCL to branch to when the user presses the preceding key. For example, the command `SetKeyboard("C",Run_calc)` causes the program to branch to the label `Run_calc` when the user presses C.

| <u>Value</u> | <u>Description</u> | <u>Value</u> | <u>Description</u> | <u>Value</u> | <u>Description</u> |
|--------------|--------------------|--------------|------------------------------|--------------|---|
| 8 | BACKSPACE | 9 | TAB | 12 | 5 on numeric keypad with NUMLOCK off |
| 13 | ENTER (or ^M) | 16 | SHIFT | 17 | CTRL |
| 18 | ALT | 19 | PAUSE (or CTRL + NUMLOCK) | 20 | CAPS LOCK |
| 27 | ESCAPE | 32 | SPACEBAR | 33 | PGUP |
| 34 | PGDN | 35 | END | 36 | HOME |
| 37 | LEFTARROW | 38 | UPARROW | 39 | RIGHTARROW |
| 45 | INSERT | 40 | DOWNARROW | 44 | PRINTSCREEN |
| 49 | 1 | 46 | DELETE | 48 | 0 |
| 52 | 4 | 50 | 2 | 51 | 3 |
| 55 | 7 | 53 | 5 | 54 | 6 |
| 65 | A | 56 | 8 | 57 | 9 |
| 68 | D | 66 | B | 67 | C |
| 70 | F | 69 | E | 72 | H |
| 73 | I | 71 | G | 75 | K |
| 76 | L | 74 | J | 78 | N |
| 79 | O | 77 | M | 81 | Q |
| 82 | R | 80 | P | 84 | T |
| 85 | U | 83 | S | 87 | W |
| 90 | Z | 86 | V | 89 | Y |
| | | 88 | X | 97 | Numeric key pad 1 |
| | | 96 | Numeric key pad 0 | | |

| | | | | | |
|-----|---|-----|---|-----|---|
| 98 | Numeric key pad 2 (NUMLOCK must be on) | 99 | Numeric key pad 3 (NUMLOCK must be on) | 100 | Numeric key pad 4 (NUMLOCK must be on) |
| 101 | Numeric key pad 5 (NUMLOCK must be on) | 102 | Numeric key pad 6 (NUMLOCK must be on) | 103 | Numeric key pad 7 (NUMLOCK must be on) |
| 104 | Numeric key pad 8 (NUMLOCK must be on) | 105 | Numeric key pad 9 (NUMLOCK must be on) | 106 | Numeric key pad * |
| 107 | Numeric key pad + | 109 | Numeric key pad - | 110 | Numeric key pad . (NUMLOCK must be on) |
| 111 | Numeric key pad / | 112 | Function Key F1 | 113 | Function Key F2 |
| 116 | Function Key F5 | 114 | Function Key F3 | 115 | Function Key F4 |
| 119 | Function Key F8 | 117 | Function Key F6 | 118 | Function Key F7 |
| 122 | Function Key F11 | 120 | Function Key F9 | 121 | Function Key F10 |
| 125 | Function Key F14 | 123 | Function Key F12 | 124 | Function Key F13 |
| 144 | NUM LOCK | 126 | Function Key F15 | 127 | Function Key F16 |
| | | 145 | SCROLL LOCK | | |

The following key codes apply to US keyboards only:

| | |
|-----|--|
| 186 | Colon/semi-colon |
| 187 | Plus/equal |
| 188 | Less than/comma |
| 189 | Underscore/hyphen |
| 190 | Greater than/period |
| 191 | Question/slash |
| 192 | Tilde/backwards single quote |
| 219 | Left curly brace/left square brace |
| 220 | Pipe symbol/backslash |
| 221 | Right curly brace/right square bracket |
| 222 | Double quote/single quote |

Virtual Key Numbers Table

Remarks:

When PiXCL encounters a [SetKeyboard](#) command in your program, it does not immediately branch anywhere. Instead, it waits until it encounters a [WaitInput\(\)](#) command (which causes the program to pause indefinitely for user input) and the user presses a specified key. Only then does control transfer to the [Label](#) associated with that key.

The syntax above shows only three keys in the [SetKeyboard](#) list, but you can actually include as many keys as you want in the list.

A [SetKeyboard](#) command remains in effect until any of the following occurs:

- You use another [SetKeyboard](#) command.
- You use [SetKeyboard\(\)](#) without any parameters to reset the keyboard.
- The program ends.

When you place a 3-D button in a window using the [Button](#) command, PiXCL automatically provides mouse support for it. If you want the user to be able to select the button using the keyboard, though, you must use [SetKeyboard](#) (see the second example).

Examples:

The following example puts the message "Press E to end the program" on the screen, and then pauses the program indefinitely

until you press **e** or **E**. When you press either key, the program ends.

```
    DrawText(1,1,"Press E to end the program")
    SetKeyboard("E",End_it,"e",End_it)
    WaitInput()
End_it:
    End
```

A [SetKeyboard](#) command has to be very soon followed by a [WaitInput\(\)](#) command. If you place a [SetKeyboard](#) in a code section that does many other processing, especially [Vid*](#) commands, you may get odd effects, including application crashes. This is because of the way that PiXCL processes a selected key stroke, by looking at the position of the [SetKeyboard](#) command in the script.

This next example places two buttons on the screen. When you click on a button with the mouse, or press the key associated with the button, the program runs the named application. On the other hand, if you press F1, the program displays a message box with some simple help text.

```
{Draw buttons}
    Button(22,13,95,35,"&Notepad",Run_Notepad,
          22,43,95,65,"&Calc",Run_Calc)

{Set up keyboard support for buttons}
    SetKeyboard("N",Run_Notepad,
               "n",Run_Notepad,
               "C",Run_Calc,
               "c",Run_Calc,
               112,Help_Box)      {112=virtual key for F1}

Wait_for_input:
    WaitInput()

Run_Notepad:
    Run("NOTEPAD.EXE")
    Goto Wait_for_input

Run_Calc:
    Run("CALC.EXE")
    Goto Wait_for_input

Help_Box:
    MessageBox(OK,1,NOICON,
               "Pick a button to run the named application",
               "Help box",Ignore)
    Goto Wait_for_input
```

Related Commands:

[Button](#), [SetMenu](#), [SetMouse](#), [SetCtrlMouse](#), [SetRightMouse](#), [SetShiftRightMouse](#),
[WaitInput](#)

SetListBitmapPixel

The [GetPixel](#) command retrieves a pixel coordinate RGB value from the client area, which is not necessarily a 24 bit color mode as set in the video driver. The [SetListBitmapPixel](#) command sets a pixel RGB value at a coordinate in a bitmap loaded into the PiXCL image list.

Syntax: [SetListBitmapPixel\(Handle,Line,Pixel,Index,R,G,B,Result\)](#)

Parameters:

| | |
|----------------------------|--|
| Handle | The handle of a loaded bitmap, as returned by SetCurrentBitmap , TuneImage and others. |
| Line,Pixel | The coordinate of the pixel to set the RGB values. |
| Index | The pixel index value for paletted bitmaps. Ignored for a 24-bit image pixel. |
| R,G,B | The Red, Green and Blue colour values. Set these to -1 if not required in a paletted bitmap. Values in the range 0-255 will set the colour map of the target bitmap. |

Related Commands;

[GetListBitmapPixel](#) [GetPixel](#) [SetCurrentBitmap](#)

SetLocalTime, SetSystemTime

System time, actually Greenwich Mean Time or GMT, and Local system time, based on the time zone to which your PC is set can be accessed and if desired, reset. Both commands have the same arguments.

Syntax: *SetLocalTime(Year, Month, DayOfWeek, DayOfMonth, Hour, Minutes)*
SetSystemTime(Year, Month, DayOfWeek, DayOfMonth, Hour, Minutes)

PIXCL 5

SetLocalTime(Year, Month, DayOfWeek, DayOfMonth, Hour, Minutes, Seconds)
SetSystemTime(Year, Month, DayOfWeek, DayOfMonth, Hour, Minutes, Seconds)

Parameters:

| | |
|-------------------|---------------------------------|
| <i>Year</i> | Four digit integer, e.g. 2001 |
| <i>Month</i> | Range is 1 to 12 |
| <i>DayOfWeek</i> | Range is 0=Sunday to 6=Saturday |
| <i>DayOfMonth</i> | Range is 1 to 31 |
| <i>Hour</i> | Range 1 to 23 |
| <i>Minutes</i> | Range 0 to 59 |
| <i>Seconds</i> | Range 0 to 59 |

Related Commands:

[GetLocalTime](#), [GetSystemTime](#), [GetTimeZone](#), [TimeToASCII](#)

SetMapMode

Get the current client area mapping mode.

Syntax: [SetMapMode\(TEXT | ISOTROPIC | ANISOTROPIC\)](#)

Parameter:

| | |
|-----------------------------|------------------------|
| TEXT | The default mode. |
| ISOTROPIC | Origin is top left. |
| ANISOTROPIC | Origin is bottom left. |

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetWindowExtent](#) [SetWindowOrigin](#) [SetViewportExtent](#) [SetViewportOrigin](#)

SetMenu

This command is used to create your own custom menus. When the program is pausing for input and the user selects a menu item, the program branches to the label associated with that menu item. A menu command from a PiXCL 4 application is completely compatible with PiXCL 5, that adds the capability for child pop up menus.

Syntax:

SetMenu()

Or

PIXCL 4

```
SetMenu(Top1$,IGNORE/Label,  
ItemA$, Label,  
ItemB$, Label,  
SEPARATOR,  
ItemC$, Label,  
ENDPOPOP,  
Top2$, Label,  
.  
.  
ENDPOPOP)
```

PIXCL 5

```
SetMenu(Top1$,IGNORE|Label,  
ItemA$, Label,  
ItemB$, Label,  
ItemX$, CHILDDPOPUP,  
ItemX1$, Label,  
ItemX2$, Label,  
ENDCHILDDPOPUP,  
SEPARATOR,  
ItemC$, Label,  
ENDPOPOP,  
Top2$, Label,  
.  
.  
ENDPOPOP)
```

Parameters:

Top1\$, Top2\$

The items that are to appear on the main or top-level menu bar. You can have as many items as you like on the main menu bar. (If you have more than a single line's worth, Windows will extend the menu bar to a second line and beyond.)

ItemA\$, ItemB\$,

The items that are to appear within a popup *ItemC\$* menu, also called a *pull-down menu*. You can have as many items as you like within a popup menu.

Label

A label you want PiXCL to branch to when the user selects a menu item.

IGNORE

Tells PiXCL not to do anything when the menu item is selected. When IGNORE follows an item on the main menu bar, PiXCL displays the item's popup menu, provided you've defined one.

SEPARATOR

Divides the items in a popup menu into groups.

ENDPOPOP

Ends a popup menu. In addition, this token is always the last argument in a *SetMenu* command.

PIXCL 5

CHILDDPOPUP

Token that indicates the next items are a child popup menu.

ENDCHILDDPOPUP

Token that completes the definition of the child popup menu.

Remarks:

A top-level menu consists of one or more items--*Top1\$, Top2\$, Top3\$*, and so on. Below each top-level menu item is a popup

menu. You can have one or more items within a popup menu--*ItemA\$, ItemB\$, ItemC\$,* and so on.

When you define a menu in PiXCL, you define it sequentially. You begin by setting up the first top-level menu item and its popup menu. You then set up the second top-level menu item and its popup, and so on.

Following each top-level menu item (*Top1\$*) and popup menu item (*ItemA\$*) is a label you want the program to branch to when a menu item is selected. If you don't want the program to branch anywhere, use the IGNORE token instead; the IGNORE token is most often used following a top-level menu item when all you want PiXCL to do is show the item's popup menu. (The IGNORE token is recognized even if you've created an IGNORE label.)

When PiXCL encounters a *SetMenu* command in your program, it does not immediately branch anywhere. Instead, it waits until it encounters a *WaitInput()* command (which causes the program to pause indefinitely for user input) and the user selects a menu item. Only then does control transfer to the *Label* associated with that item.

Here are some conventions you may want to follow when creating PiXCL menus in order to give the user additional information about the items within the menu:

- An *underlined letter* in a menu item indicates that the letter can be used to select the menu item. You create underlined letters by placing an & in front of the letter you want underlined. For example, the parameter "&Notepad" underlines the letter N in the Notepad menu item. When a letter is underlined, it is known as a *mnemonic*. To select an item on the top-level menu using its mnemonic, you press ALT+mnemonic, for example ALT+N. Once a popup menu appears, you can select an item within it by pressing that item's mnemonic alone.
- By placing an *exclamation point* at the end of a top-level menu item, you can indicate that no pop-up menu will appear when the user selects the item.
- By using a *separator*, you can divide items within a popup menu into groups. You can have as many separators as you like within a popup menu.

A *SetMenu* command remains in effect until any of the following occurs:

- You use another *SetMenu* command.
- You use *SetMenu* without any parameters to eliminate the menu from the window.
- The program ends.

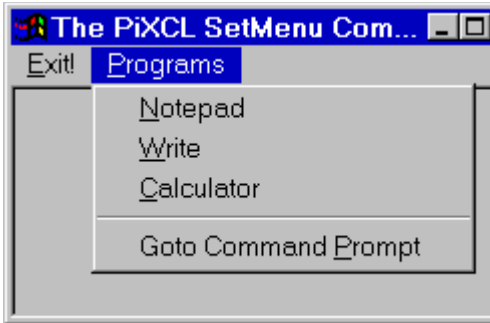
You can add hot-key text--for example, *Ctrl+N*--to a menu item by preceding the text with a tab character. For example, here's some code that creates a menu item of the form

```
"Notepad Ctrl+N":
```

```
Chr(9, Tab$)  
Notepad$ = "&Notepad" + Tab$  
Notepad$ = Notepad$ + "Ctrl+N"
```

You can then use the *Notepad\$* variable when defining a menu item in *SetMenu*. Be aware that PiXCL does not automatically provide keyboard support for hot-key text you assign. You must provide the support yourself using the *SetKeyboard* command.

Example:



A menu created with SetMenu.

This example creates the simple menu shown left. As you can see, the menu has two top-level menu items, Programs and Exit!. (Because Exit! does not lead to a popup menu, it includes an exclamation point.) When you select an item from Programs' popup menu, the program branches to the associated label. For example, when you select *Goto Command Prompt*, the program branches to the label Interpret, where it launches a copy of the command interpreter, CMD.EXE.

```
{Set up the menu}
SetMenu("&Exit!", Leave,
        ENDDOPOPUP,
        "&Programs", IGNORE,
        "&Notepad", Run_Notepad,
        "&Write", Run_Notepad,
        "&Calculator", Run_Notepad,
        SEPARATOR,
        "Goto Command &Prompt", Run_Notepad,
        ENDDOPOPUP)
Wait_for_input:
    WaitInput()

Run_Notepad:
    Run("NOTEPAD.EXE")
    Goto Wait_for_input

Run_Write:
    Run("WRITE.EXE")
    Goto Wait_for_input

Run_Calc:
    Run("CALC.EXE")
    Goto Wait_for_input

Interpret:
    DirGetSystem(WindowsSystem$)
    Cmd$=WindowsSystem$+"\cmd.exe"
    Run(Cmd$)
    Goto Wait_for_input

Shutdown:
    End
```

Related Commands:

[GetMenuStatus](#), [SetKeyboard](#), [SetPopupMenu](#), [WaitInput](#)

SetMouse

This command lets you get mouse input. When the program is pausing for input and the user clicks the left mouse button within a specified rectangular region, the program branches to the label associated with that region.

Syntax:

`SetMouse()`

or

```
SetMouse(Region1_x1,Region1_y1,Region1_x2,Region1_y2,Label,x,y,  
         Region2_x1,Region2_y1,Region2_x2,Region2_y2,Label,x,y,  
         .  
         .  
         Regionn_x1,Regionn_y1,Regionn_x2,Regionn_y2,Label,x,y)
```

Parameters:

| | |
|------------------------------------|---|
| <code>Regionx_x1,Regionx_y1</code> | The upper-left corner of a rectangular mouse hit-testing region. |
| <code>Regionx_x2,Regionx_y2</code> | The lower-right corner of a rectangular mouse hit-testing region. |
| <code>Label</code> | The label you want to branch to when the user clicks the mouse within the mouse hit-testing region (as defined by the previous four arguments). |
| <code>x,y</code> | The coordinates of the mouse pointer, as measured from the upper-left corner of the window. |

Remarks:

To get mouse input, you must set up rectangular areas in the window known as *mouse hit-testing regions*. When the user clicks the mouse within a hit-testing region, the program branches to the label associated with that region and saves the mouse pointer coordinates in the two variables that follow, *x* and *y*.

When PiXCL encounters a `SetMouse` command in your program, it does not immediately branch anywhere. Instead, it waits until it encounters a `WaitInput()` command (which causes the program to pause indefinitely for user input) and the user clicks the mouse within a hit-testing region. Only then does control transfer to the *Label* associated with that region.

You can have as many hit-testing regions as you like for a given `SetMouse` command. If two hit-testing regions overlap, PiXCL will branch to the label for the first one in the list.

When you place a 3-D button in a window using the `Button` command, PiXCL automatically provides mouse support for the button--you do not need to use `SetMouse` for this.

Setting the coordinate system to pixels is helpful when you want to get mouse input (see `UseCoordinates`).

Example:

The following program sets up a mouse hit-testing region corresponding to the rectangle on the left-hand side of the window. When you click the mouse within the rectangle--the region (1,1) to (300,200)--the program saves the mouse pointer coordinates in `Mouse_x` and `Mouse_y` and branches to the `Mouse_hit` label, where it draws a small black rectangle at the location of the mouse pointer.

```
{Set coordinate system to pixels}
```



```
UseCoordinates(PIXEL)

{Draw rectangle that will become hit-testing region}
  DrawRectangle(1,1,300,200)
  DrawText(30,210,"Click the mouse within the rectangle")

{Set up the mouse}
  SetMouse(1,1,300,200,Mouse_hit,Mouse_x,Mouse_y)

{Put up Exit button}
  Button(340,80,400,120,"Exit",Goodbye)

Wait_for_Input:
  WaitInput()

Mouse_hit:
  x2=Mouse_x+2
  y2=Mouse_y+2
  DrawRectangle(Mouse_x,Mouse_y,x2,y2)
  Goto Wait_for_Input

Goodbye:
  End
```

See the [DrawNumber](#) and [SetKeyboard](#) commands for other examples of [SetMouse](#).

Related Commands:

[SetDbfMouse](#) [SetCtrlMouse](#) [SetMidMouse](#) [SetRightMouse](#) [SetShftRightMouse](#)

SetPopupMenu

In Windows 95 and NT 4.0 and later, extensive use is made of popup menus that appear when the right mouse button clicked. These menus are often also called context sensitive menus, and typically provide access to most commonly used commands. PiXCL supports popup menus, and uses essentially the same syntax the [SetMenu\(\)](#) command that creates the standard menu bar.

The [SetPopupMenu](#) command would normally be used with the [SetRightMouse](#) command, but will work with any of the [SetMouse](#) commands too. It can be easily programmed to create any size popup menu at any location. When you click the mouse in the PiXCL application client area, and invoke a [SetPopupMenu\(...\)](#) function, the top left corner of the popup menu appears at the coordinates returned by the [Set\[Shift|Ctrl|Right|DbL\]Mouse](#) command.

The same syntax used in the [SetMouse](#) commands applies with some small exceptions. You can use [SEPARATOR](#) tokens, but there should be only one [ENDPOPUP](#) token. Multiple [ENDPOPUP](#) tokens will not cause errors, but are not meaningful. An [ENDPOPUP](#) after a [SEPARATOR](#) will be ignored. The [IGNORE](#) token, used in top level menu items with pull down menus, is not meaningful in a popup menu. If you try to use the [IGNORE](#) token, a popup menu will be created, but selecting this menu item will respond with a syntax error message.

You can also define accelerator keys (e.g. "&Item#1",...), just as you can with the standard [SetMenu](#) command.

[SetPopupMenu\(\)](#) Clear all popup menu commands.

or

```
SetPopupMenu(ItemA$,Label,  
            ItemB$,Label,  
            SEPARATOR,  
            ItemC$,Label,  
            ...  
            ...  
            ENDPOPUP)
```

Parameters:

[ItemA\\$, ItemB\\$,](#) The items that are to appear within a popup [Item\\$](#) menu. You can have as many items as you like within a popup menu.

[Label](#) A label you want PiXCL to branch to when the user selects a menu item.

[SEPARATOR](#) Divides the items in a popup menu into groups.

[ENDPOPUP](#) Ends a popup menu. In addition, this token is always the last argument in a [SetPopupMenu](#) command.

[CHILDPOPUP](#) **PiXCL 5:** add a child popup menu.

[ENDCHILDPOPUP](#) Terminate the definition of the child popup menus. You can use the [SEPARATOR](#) token within a child popup.

Example:

```
SetRightMouse (cx1, cy1, cx2, cy2, Test1, X, Y)  
.  
.
```

```
.  
Test1:  
    SetPopupMenu("Item #&1",Item_1,  
                SEPARATOR,  
                "Item #&2",Item_2,  
                SEPARATOR,  
                "Item #&3",Item_3,  
                "Item #&4",Item_4,  
                ENDFPOPUP)  
    Goto Wait_for_Input
```

Related Commands:

[SetMenu](#), [SetDbfMouse](#), [SetCtrlMouse](#), [SetRightMouse](#), [SetShftRightMouse](#)

SetPNGOptions

PIXCL 5 command. When you load a PNG image from disk, the options data in the file, if any, are also loaded and stored with the image in the PiXCL image list. These fields can be set or updated as well. If the disk image type is not PNG, the command has no effect, and all values return empty strings. You can get this command to work if you first set change the name of the loaded image in the list, using [RenameListImage](#).

Syntax: `SetPNGOptions(Filename$, Title$, Author$, Copyright$, Description$,
Software$, Warning$, Disclaimer$, Source$, Comment%)`

Parameters:

Filename\$ The name of a loaded file in the image list.

Note all the following MUST be string variables, or you will get a syntax error.

Title\$ The contents of the field. This may be an empty string.

Author\$ The contents of the field. This may be an empty string.

Copyright\$ The contents of the field. This may be an empty string.

Description\$ The contents of the field. This may be an empty string.

Software\$ The contents of the field. This may be an empty string.

Warning\$ The contents of the field. This may be an empty string.

Disclaimer\$ The contents of the field. This may be an empty string.

Source\$ The contents of the field. This may be an empty string.

Comment\$ The contents of the field. This may be an empty string.

Related Commands:

[GetJPGOptions](#) [GetTIFFOptions](#) [SetJPGOptions](#) [SetPNGOptions](#) [SetTIFFOptions](#) [RenameListImage](#)

SetPriority

Sets the priority of the PiXCL process or the process of an application you've launched from within PiXCL using the RunExt command.

Syntax: `SetPriority(CommandLine$,IDLE/NORMAL/HIGH,Result)`

Parameters:

CommandLine\$ A string containing the command line (filename plus optional parameters) used with the RunExt command to launch an application. If **CommandLine\$** is a null string (" "), sets the priority of the PiXCL process.

IDLE Sets the priority of the application's process to level 4, the same priority level as a screen saver.

NORMAL Sets the priority of the application's process to level 9 when the application is in the foreground and to level 7 when the application is in the background. You should use NORMAL for most applications.

HIGH Sets the priority of the application's process to level 13, the same priority level as the Task Manager. You should use this setting only when absolutely necessary.

Result If the process's priority was changed, this integer variable is assigned a value of 1. Otherwise, it is assigned a value of 0.

Remarks:

You cannot change the priority of a running process unless you've launched it from within PiXCL using the [RunExt](#) command.

Windows NT supports four priority classes for processes: idle, normal, high, and real-time. (Setting a process to real-time priority can bring down Windows NT and is therefore not included in PiXCL.) PiXCL lets you assign a priority class to an application's main process, provided you've launched the application from within PiXCL using the [RunExt](#) command. (If the application was launched in another manner--with the [Run](#) command, for example--PiXCL won't have the proper security rights to it and therefore won't be able to change the application's priority.)

If you're having trouble sending keystrokes to an application with [SendKeys](#), try raising or lowering the priority of the application's main process (or the priority of PiXCL's main process). Another useful thing to try is raising the priority of the hook thread within PiXCL; see the [SetSendKeysPriority](#) command.

Related Commands:

[RunExt](#), [SendKeys](#), [SetSendKeysPriority](#)

SetRightMouse, SetDbIRightMouse, SetShftRightMouse, SetCtrlRightMouse

These commands are like the [SetMouse](#) command, except that they work with right-mouse, control right-mouse, shift right-mouse and double right-mouse clicks.

Syntax:

[SetRightMouse\(\)](#)
[SetDbIRightMouse\(\)](#)
[SetCtrlRightMouse\(\)](#)
[SetShftRightMouse\(\)](#)

or

[SetRightMouse\(Region1_x1,Region1_y1,Region1_x2,Region1_y2,Label,x,y,
Region2_x1,Region2_y1,Region2_x2,Region2_y2,Label,x,y,
...
...
Regionn_x1,Regionn_y1,Regionn_x2,Regionn_y2,Label,x,y\)](#)

Remarks:

See the [SetMouse](#) command for an explanation of [SetRightMouse](#) and [SetShftRightMouse](#)'s arguments and behavior.

Related Commands:

[SetCtrlMouse](#), [SetMouse](#), [SetDbIMouse](#)

SetROPcode

PIXCL creates two bitmap images of the client area, one in memory and the other in the screen.

The [SetROPcode](#) commands provides a method of changing how the foreground image (in the screen or destination bitmap) is redrawn from the background image (in the memory or source bitmap), by changing the raster operation, or ROP, code. The default method is direct copy from source to destination (token [SRCCOPY](#)).

Syntax: [SetROPcode\(TOKEN\)](#)

Parameters:

| | |
|-----------------------------|---|
| BLACKNESS | Fills the destination rectangle using the color associated with index 0 in the physical palette. (This color is black for the default physical palette.) |
| DSTINVERT | Inverts the destination rectangle. |
| MERGECOPY | Merges the colors of the source rectangle with the specified pattern by using the Boolean AND operator. |
| MERGEPAIN | Merges the colors of the inverted source rectangle with the colors of the destination rectangle by using the Boolean OR operator. |
| NOTSRCCOPY | Copies the inverted source rectangle to the destination. |
| NOTSRCERASE | Combines the colors of the source and destination rectangles by using the Boolean OR operator and then inverts the resultant color. |
| PATCOPY | Copies the specified pattern into the destination bitmap. |
| PATINVERT | Combines the colors of the specified pattern with the colors of the destination rectangle by using the Boolean XOR operator. |
| PATPAINT | Combines the colors of the pattern with the colors of the inverted source rectangle by using the Boolean OR operator. The result of this operation is combined with the colors of the destination rectangle by using the Boolean OR operator. |
| SRCAND | Combines the colors of the source and destination rectangles by using the Boolean AND operator. |
| SRCCOPY | Default Value. Copies the source rectangle directly to the destination rectangle. This is the default value when a PiXCL application starts up. |
| SRCERASE | Combines the inverted colors of the destination rectangle with the colors of the source rectangle by using the Boolean AND operator. |
| SRCINVERT | Combines the colors of the source and destination rectangles by using the Boolean XOR operator. |
| SRCPAIN | Combines the colors of the source and destination rectangles by using the Boolean OR operator. |
| WHITENESS | Fills the destination rectangle using the color associated with index 1 in the physical palette. (This color is white for the default physical palette.) |

Remarks:

Setting the ROP code enables you to draw in the screen bitmap only, without drawing in the background bitmap. This can be useful to create a client area bitmap with the SRCCOPY mode that is stored in both screen and memory, then, using another mode, draw in the screen only for a few operations, then draw the background image again. This is how simple animation operations can be achieved.

If you change the ROP code to another setting, you must reset to the default SRCCOPY or another mode or the current mode remains in place.

Related Commands:

[Redraw](#) [SetDrawMode](#)

SetSendKeysPriority

Sets the priority of the PiXCL thread that handles keystroke messages initiated by SendKeys.

Syntax:

[SetSendKeysPriority](#)(LOWEST / BELOW_NORMAL / NORMAL / ABOVE_NORMAL / HIGHEST)

Parameters:

| | |
|------------------------------|---|
| LOWEST | The hook thread's priority should be two less than the PiXCL process's priority. |
| BELOW_NORMAL | The hook thread's priority should be one less than the PiXCL process's priority. |
| NORMAL | The hook thread's priority should be the same as the PiXCL process's priority. This is the default. |
| ABOVE_NORMAL | The hook thread's priority should be one more than the PiXCL process's priority. |
| HIGHEST | The hook thread's priority should be two more than the PiXCL process's priority. |

Remarks:

When you send keystrokes to other applications using [SendKeys](#), PiXCL sets up a hook to monitor the flow of messages through the system. This hook is controlled by a second thread of execution, separate from the thread that controls the main PiXCL program.

The priority of the thread is set relative to the PiXCL process's priority. The default setting is [NORMAL](#), which is appropriate for sending keystrokes to most 32-bit Windows applications. In certain cases, though, you may need to raise (or lower) the priority of the hook thread in order to process messages. This is particularly true when sending keystrokes to 16-bit Windows apps if it doesn't appear to accept them. Another good time is when PiXCL appears to hang during a [SendKeys](#) command.

This setting has no effect unless you're using the [SendKeys](#) command. If you want to control the priority of an application, start the application from within PiXCL using [RunExt](#). You can also change the priority of an application started with [RunExt](#) by using the [SetPriority](#) command.

Related Commands:

[SendKeys](#), [SetPriority](#), [Run](#), [RunExt](#)

SetTextSpacing

The [SetTextSpacing](#) command sets a new intercharacter spacing. The effect is immediately available with all of the text drawing commands.

Syntax: [SetTextSpacing\(*Spacing*\)](#)

Parameter:

[Spacing](#) Has to be an integer variable. The desired new spacing between characters.

Remarks:

Text spacing can be different if required in the foreground and background. Use the [SetDrawMode](#) command before [GetTextSpacing](#) and [SetTextSpacing](#).

Releted Commands:

[DrawText](#) [DrawNumber](#) [SetDrawMode](#) [GetTextSpacing](#)

SetTIFOptions

PIXCL 5 command. When you load a TIF image from disk, the options data in the file, if any, are also loaded and stored with the image in the PiXCL image list. These fields can be set or updated as well. If the disk image type is not TIF, the command has no effect, and all values return an empty string. You can get this command to work if you first set change the name of the loaded image in the list, using [RenameListImage](#).

Syntax: `SetTIFOptions(Filename,$,Artist,$,Description,$,Software,$,HostComputer,$,DocName,$)`

Parameters:

Filename\$ The name of a loaded file in the image list.

Note all the following MUST be string variables, or you will get a syntax error.

Artist\$ The contents of the field. This may be an empty string.

Description\$ The contents of the field. This may be an empty string.

Software\$ The contents of the field. This may be an empty string.

HostComputer\$ The contents of the field. This may be an empty string.

DocName\$ The contents of the field. This may be an empty string.

Related Commands:

[GetJPGOptions](#) [GetPNGOptions](#) [GetTIFOptions](#) [SetJPGOptions](#) [SetPNGOptions](#) [RenameListImage](#)

SetupColorMatching

PIXCL 5 command. For Windows 98 / 2000 based systems, both of which have the ICC colour matching libraries installed, colour matching can be set up with the dialog that this command displays.

Syntax: SetupColorMatching(*Profile*\$, *Display*\$, *Printer*\$, **Intent_TOKEN**, *Result*)

Parameters:

Profile\$

Display\$

Printer\$

Intent_TOKEN

PERCEPTUAL

SATURATION

RELATIVE_COLORIMETRIC

ABSOLUTE_COLORIMETRIC

Result

1 if the operation succeeded, otherwise 0.

Related Commands:

SetVECdrawParams

Ascii **VEC**tor files can have coordinates expressed in integer or floating point format, and refer to a region with a different coordinate space than the PiXCL client area where you want to display an image. The raw VEC file coordinate space can be transformed into the display coordinate space by setting the gain and offset for the X and Y axes. You are responsible for calculating the gains and offsets, either externally with hand coding, or have your PiXCL program calculate them for you.

Syntax: [SetVECdrawParams\(Xgain, Ygain, Xoffset, Yoffset, x1, y1, x2, y2\)](#)

Parameters:

| | |
|----------------------------------|--------------------------------------|
| Xgain, Ygain | Coordinate gain * 100 |
| Xoffset, Yoffset | Coordinate offset * 100 |
| x1, y1, x2, y2 | Client area region in which to draw. |

Remarks:

The parameters set by this command are used by all subsequent [DrawVEC](#) commands, until a new [SetVECdrawParams](#) command is issued, or the program exits.

Related Commands:

[DrawVEC commands](#)

SetViewportExtent

Set the X and Y extent of the viewport in the client area.

Syntax: [SetViewportExtent\(Xextent, Yextent\)](#)

Parameters:

[Xextent](#) The viewport extent in the X-axis. This number can be negative.

[Yextent](#) The viewport extent in the Y-axis. This number can be negative.

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowExtent](#)
[SetWindowOrigin](#) [SetViewportOrigin](#)

SetViewportOrigin

Set the X and Y origin of the viewport in the client area.

Syntax: [SetViewportOrigin\(Xorigin, Yorigin\)](#)

Parameters:

[Xorigin](#) The viewport origin in the X-axis.

[Yorigin](#) The viewport origin in the Y-axis.

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowExtent](#)
[SetWindowOrigin](#) [SetViewportExtent](#)

SetWaitMode

This command is obsolete in PiXCL 4.0 and later, and has no effect. It is maintained merely for compatibility with earlier versions. In Windows 3.1, it controlled how PiXCL behaves after a Run command starts another application and PiXCL encounters a [WaitInput\(\)](#) command in your script. If you need to pause a PiXCL program until another PiXCL application is complete, use the [PXLResume](#) or [PXLResumeAt](#) command.

Syntax: [SetWaitMode\(NULL/FOCUS\)](#)

Parameters:

| | |
|-----------------------|---|
| NULL | Causes WaitInput to behave in the normal way. That is, WaitInput with an argument pauses PiXCL a specified number of milliseconds, and WaitInput without an argument pauses PiXCL indefinitely while it waits for user input. SetWaitMode(NULL) is the default. |
| FOCUS | Causes PiXCL to pause until the focus returns when it encounters a WaitInput(1) command. |

Related Commands:

[WaitInput](#), [PXLResume](#), [PXLResumeAt](#)

SetWindow

Maximizes, minimizes, or restores the PiXCL window. This is an obsolete command provided for compatibility with earlier versions of PiXCL. The [WinShow](#) command is more powerful, and should be used in preference.

Syntax: [SetWindow\(MAXIMIZE/MINIMIZE/RESTORE\)](#)

Parameters:

[MAXIMIZE](#) Maximizes the PiXCL window.

[MINIMIZE](#) Minimizes the PiXCL window.

[RESTORE](#) Restores the PiXCL window.

Remark:

The [SetWindow](#) command is provided only for compatibility with previous releases of PiXCL. The [WinShow](#) command is a more powerful replacement for [SetWindow](#); it lets you hide, unhide, minimize, maximize, or restore any application window, including PiXCL's.

Example:

This program uses the [SetWindow](#) command to maximize, minimize, and restore the PiXCL window in various ways, pausing for one second in between [SetWindow](#) commands.

```
SetWindow(MAXIMIZE)
WaitInput(1000)
SetWindow(RESTORE)
WaitInput(1000)
SetWindow(MINIMIZE)
WaitInput(1000)
SetWindow(RESTORE)
WaitInput()
```

Related Commands

[WinClose](#), [WinExist](#), [WinGetLocation](#), [WinLocate](#), [WinSetActive](#), [WinShow](#)

SetWindowExtent

Set the X and Y extent of the client area.

Syntax: [SetWindowExtent\(Xextent, Yextent\)](#)

Parameters:

[Xextent](#) The extent in the X-axis. This number can be negative.

[Yextent](#) The extent in the Y-axis. This number can be negative.

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowOrigin](#)
[SetViewportExtent](#) [SetViewportOrigin](#)

SetWindowOrigin

Set the X and Y origin of the window in the client area.

Syntax: [SetWindowOrigin\(Xorigin, Yorigin\)](#)

Parameters:

[Xorigin](#) The window origin in the X-axis.

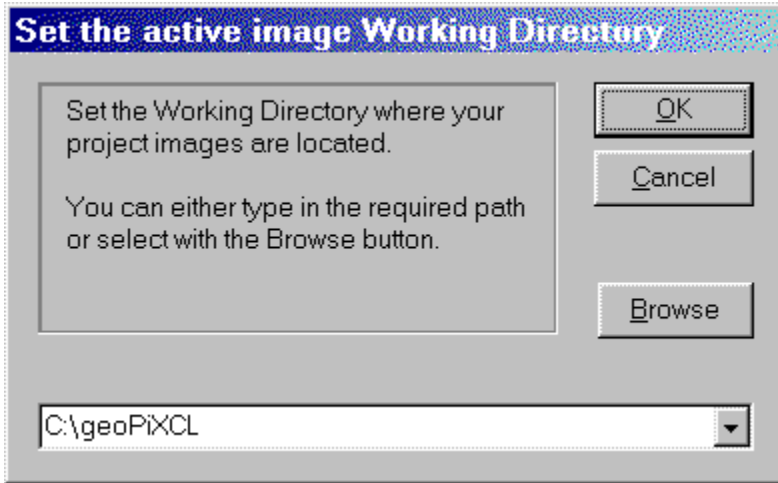
[Yorigin](#) The window origin in the Y-axis.

Related Commands:

[GetMapMode](#) [GetWindowExtent](#) [GetWindowOrigin](#) [GetViewportExtent](#) [GetViewportOrigin](#) [SetMapMode](#) [SetWindowExtent](#)
[SetViewportExtent](#) [SetViewportOrigin](#)

SetWorkingDirBox

The PiXCL MDI Editor and the geoPiXCL display engine both have a dialog to select the current working directory for project or application files. The same dialog is available using the [SetWorkingDirBox](#) command. The default title and text are shown below, and can be changed by setting string variables in the command. The contents of the combobox are created from entries in the Registry, if they exist. See the **Remarks** below for more information.



Syntax: [SetWorkingDirBox](#)(*Title\$, StaticText\$, RegistryHandle, Selection\$, Result*)

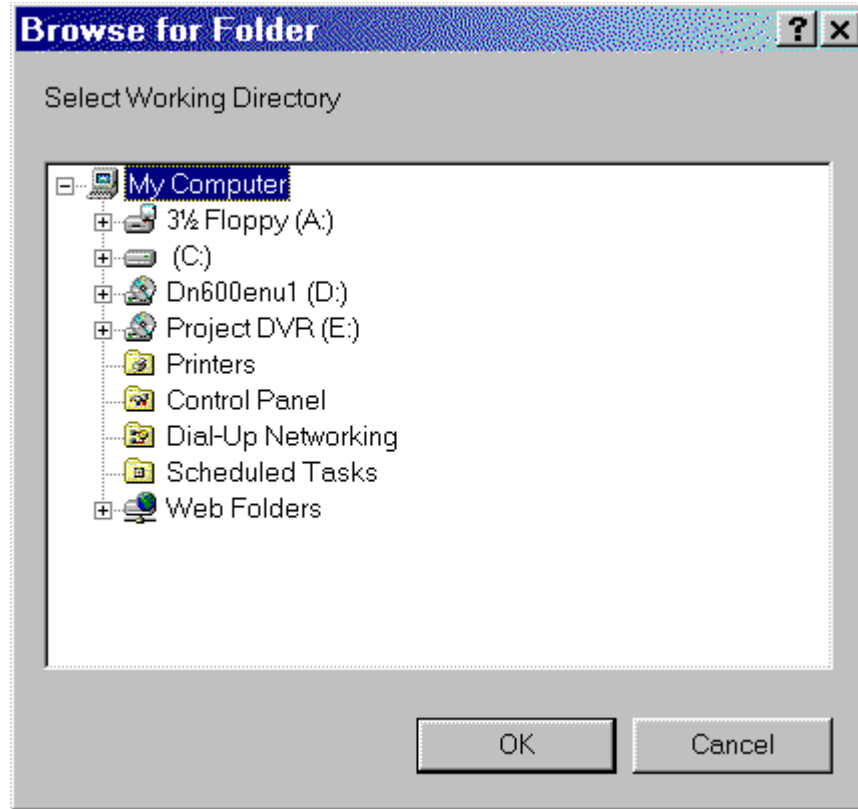
Parameters:

- [Title\\$](#) A user defined title string. Leave this null to display the default “[Set the active image working directory](#)”.
- [StaticText\\$](#) A user defined string if the default is not suitable. Leave this null to display the default.
- [RegistryHandle](#) A handle returned by [RDBOpenKey](#). This handle has to be closed by a call to [RDBCcloseKey](#) when this command returns.
- [Selection\\$](#) The selected directory returned from the dialog.
- [Result](#) 1 if the operation was successful, otherwise 0.

Remarks:

Pressing the browse button causes the dialog (right) to appear. You can select the desired directory, and when OK is clicked, the selected directory string appears in the previous dialog edit control.

Pressing Cancel has no effect on the content of the previous dialog edit control.



The current working directory is stored in the Registry. The recommended key for application parameters is

HKEY_CURRENT_USER\Software\Company\Product\Settings

Where **Company** is the name of your organisation, and **Product** is the name of the package. For example, if you look in the Registry with **regedit** or **regedt32**, you will see **VYSOR Integration Inc** and **PIXCL MDI Editor** and **geoPIXCL Remote Sensing** if you purchased the geoPIXCL product.

Additional string values are written into the **Settings** key as follows.

CurDir which is where the selected working directory is stored.

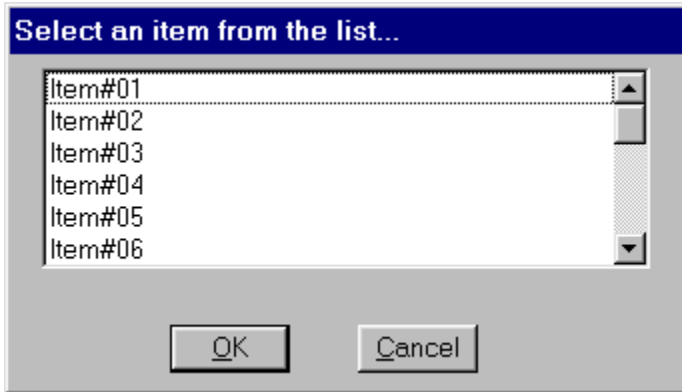
PrevDir#1 – PrevDir#4 that are created and filled as you add more selections. These previous working directories are written into the drop-down list of the combobox.

Related Commands:

[RDBCcloseKey](#) [RDBOpenKey](#)

ShellAbout

Windows has an about box built-in that can be called by the [ShellAbout](#) command. A dialog example is shown below, with the Windows 95 large icon. If you run Windows NT the icon will be different. You can also select one of the icons built into PiXCL if you want.



Syntax: [ShellAbout](#)(*Title\$*,*Info\$*,*ICON*)

Parameters:

| | |
|-----------------|--|
| <i>Title\$</i> | Text displayed in the title bar of the Shell About dialog box and on the first line of the dialog box after the text "Microsoft". If the text contains a "#" separator, dividing it into two parts, the function displays the first part in the title bar, and the second part starting on the same line as, and to the right of the text "Microsoft". Only one line of text after the "#" is accepted, so keep the text brief, or set it to NULL or a space character. |
| <i>Info\$</i> | Text that the function displays in the dialog box after the Microsoft version and copyright information. |
| <i>ICON</i> | The same icons that the MessageBox command uses are available in the ShellAbout command. Some of these (e.g. QUESTION) would usually not be used. |
| INFORMATION | Displays an icon consisting of a lowercase i in a circle. |
| EXCLAMATION | Displays an exclamation-point icon. |
| QUESTION | Displays a question-mark icon. |
| STOP | Displays a stop sign icon. |
| APP | Displays a generic application logo that looks like a small dialog box. |
| WINLOGO | Displays the large Windows 95 or Windows NT logo. |
| ICON01 - ICON19 | Displays one of the icons built into PiXCL. These are the same tokens that are used in the DrawIcon and DrawIconFile commands.. |

Related Commands:

[AboutPiXCL](#) [AboutUser](#) [MessageBox](#)

SetX1Mouse, SetShftX1Mouse, SetCtrlX1Mouse, SetDbIX1Mouse

These commands are like the [SetMouse](#) command, except that they work with X1- mouse, control X1-mouse, shift and double X1 mouse clicks. All mouse commands use the same syntax. You must have a Microsoft Intellimouse™ or Explorer™ or Optical Explorer™ or equivalent installed on your system for these commands to function.

Windows 2000 supports the X buttons, and Windows 98 also if an Explorer™ or Optical Explorer™ mouse is installed.

Syntax:

[SetX1Mouse\(\)](#)
[SetShftX1Mouse\(\)](#)
[SetCtrlX1Mouse\(\)](#)
[SetDbIX1Mouse\(\)](#)

or

[SetX1Mouse\(Region1_x1,Region1_y1,Region1_x2,Region1_y2,Label,x,y,
Region2_x1,Region2_y1,Region2_x2,Region2_y2,Label,x,y,
. . .
Regionn_x1,Regionn_y1,Regionn_x2,Regionn_y2,Label,x,y\)](#)

Remarks:

See the [SetMouse](#) command for an explanation of [SetX1Mouse](#)'s arguments and behavior.

If a [SetX1Mouse](#) command is already in effect and the user double-clicks the mid mouse button, PiXCL will respond to the [SetX1Mouse](#) command first before responding to the [SetDbIX1Mouse](#) command.

Related Commands:

[SetX2Mouse](#)

SetX2Mouse, SetShftX2Mouse, SetCtrlX2Mouse, SetDbIX2Mouse

These commands are like the [SetMouse](#) command, except that they work with X2- mouse, control X2-mouse, shift and double X2 mouse clicks. All mouse commands use the same syntax. You must have a Microsoft Intellimouse™ or Explorer™ or Optical Explorer™ or equivalent installed on your system for these commands to function.

Windows 2000 supports the X buttons, and Windows 98 also if an Explorer™ or Optical Explorer™ mouse is installed.

Syntax:

[SetX2Mouse\(\)](#)
[SetShftX2Mouse\(\)](#)
[SetCtrlX2Mouse\(\)](#)
[SetDbIX2Mouse\(\)](#)

or

[SetX2Mouse\(Region1_x1,Region1_y1,Region1_x2,Region1_y2,Label,x,y,
Region2_x1,Region2_y1,Region2_x2,Region2_y2,Label,x,y,
.
.
Regionn_x1,Regionn_y1,Regionn_x2,Regionn_y2,Label,x,y\)](#)

Remarks:

See the [SetMouse](#) command for an explanation of [SetX2Mouse](#)'s arguments and behavior.

If a [SetX2Mouse](#) command is already in effect and the user double-clicks the mid mouse button, PiXCL will respond to the [SetX2Mouse](#) command first before responding to the [SetDbIX2Mouse](#) command.

Related Commands:

[SetX1Mouse](#)

ShowConsole

PIXCL 5 Command. A DOS-like console window can be created to, for example, log results of processes. When the console is no longer required, it is deleted by the [FreeConsole](#) command.

Syntax: [ShowConsole\(x1,y1,x2,y2,Result\)](#)

Parameters:

[x1,y1,x2,y2](#) The screen position for the console window.

[Result](#) 1 if the console was created, otherwise 0.

Related Commands:

[ReadConsole](#) [FreeConsole](#)

SHPAddAttribute

geoPIXCL command. When a new SHP/SHX file set is created, the **DBF** file has no attributes (also referred to as fields) associated with it. These attributes **MUST ABSOLUTELY** be added before you proceed any further. At present you can't add more attributes to a **DBF** file that has had any records added to it.

You can set as many attributes as you want for a particular **DBF** file, and ALL the records subsequently written will have the same attributes per record. To create a valid Shape file set, you should add attribute values as you add the shape coordinates with the SHPCreateObject or SHPCreateSimpleObject commands.

Syntax: *SHPAddAttribute(ShapeBaseName\$, AttributeName\$, Type_TOKEN, Width, DecimalPlaces, Result)*

Parameters:

| | |
|------------------------|--|
| <i>ShapeBaseName\$</i> | The full path. The extension SHP or SHX or DBF can be used if desired. |
| <i>AttributeName\$</i> | The name of the new field, maximum 10 characters. Longer names will be truncated. |
| <i>Type_TOKEN</i> | One of STRING , INTEGER or DOUBLE . |
| <i>Width</i> | The width in characters of a STRING field, or the maximum number of digits for INTEGER and DOUBLE types, to a maximum of 8. |
| <i>DecimalPlaces</i> | The number of decimal places for DOUBLE types. Set to 0 for integer and string types. |
| <i>Result</i> | The number of the field created starting at 0 if the operation was successful, otherwise -1. |

Related Commands:

[SHPWriteAttribute](#) [SHPReadAttribute](#)

SHPCreateSimpleObject

geoPIXCL command. A new Shape object is created with the parameters supplied, and written to the end of the specified SHP/SHX file set.

Syntax: SHPCreateSimpleObject(*ShapeBaseName*\$, *ShapeType_TOKEN*,
Vertices, *Xarray*&[*StartIndex*], *Yarray*&[*StartIndex*], *Zarray*&[*StartIndex*],
Result)

Parameters:

| | |
|-------------------------|---|
| <i>ShapeBaseName</i> \$ | The full path. No extension is needed, as the SHP and SHX extensions are added automatically. |
| <i>ShapeType_TOKEN</i> | The shapefile type, being one of the following. POINT, ARC, POLYGON, MULTIPOINT, POINTZ, ARCZ, POLYGONZ, MULTIPOINTZ, POINTM, ARCM, POLYGONM, MULTIPOINTM, MULTIPATCH |
| <i>Vertices</i> | The number of vertices in the X, Y, Z arrays. |
| <i>X Y Zarray</i> & | Previously created arrays of at least <i>Vertices</i> elements. |
| <i>StartIndex</i> | The element start index from which the vertex list is read. This will be generally be 0 but does not have to be. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[SHPCreate](#) [SHPCGetInfo](#) [SHPCWriteAttribute](#)

SHPFileCreate

geoPiXCL command. Create an empty ESRI ArcView™ Shape SHP/SHX/DBF fileset.

Syntax: *SHPFileCreate(ShapeBaseName\$, ShapeType_TOKEN, Result)*

Parameters:

ShapeBaseName\$ The full path. No extension is needed, as the **SHP, SHX** and **DBF** extensions are added automatically.

ShapeType_TOKEN The shapefile type, being one of the following.
POINT, ARC, POLYGON, MULTIPOINT,
POINTZ, AR CZ, POLYGONZ, MULTIPOINTZ,
POINTM, AR CM, POLYGONM, MULTIPOINTM,
MULTIPATCH

Result The number of files created. This should always be 3 if the operation was successful.

Related Commands:

[SHPFileGetInfo](#) [SHPWriteAttribute](#)

SHPFileGetInfo

geoPIXCL command. Reads the specified SHP/SHX file, and returns an information summary. The files are opened, read and closed by this command.

Syntax: *SHPFileGetInfo(ShapeBaseName\$, Entities, ShapeType, FloatArray& [Size], Result)*

Parameters:

ShapeBaseName\$ The full path. No extension is needed, as the **SHP**, **SHX** and **DBF** extensions are added automatically.

Entities The number of objects in the shape file set. These are referenced from 0.

ShapeType The type of objects in the shape file set. Possible values are

- 1 = **POINT**,
- 3 = **ARC**,
- 5 = **POLYGON**,
- 8 = **MULTIPOINT**,
- 11 = **POINTZ**,
- 13 = **ARCZ**,
- 15 = **POLYGONZ**,
- 18 = **MULTIPOINTZ**,
- 21 = **POINTM**,
- 23 = **ARCM**,
- 25 = **POLYGONM**,
- 28 = **MULTIPOINTM**
- 31 = **MULTIPATCH**

FloatArray& [Size] An array of at least 8 elements that holds
in [0-3] minimum bounds X,Y,Z, M (0.0 for other than Measure types), and
in [4-7] maximum bounds X,Y,Z, M (0.0 for other than Measure types).

Result 1 if the operation was successful, otherwise 0 e.g. if the file cannot be found or is not of the specified ShapeType.

Related Commands:

[SHPFileCreate](#) [SHPFileGetType](#)

SHPFileGetType

geoPIXCL command. Reads the specified SHP/SHX file, and returns the number and type of the entities. The files are opened, read and closed by this command.

Syntax: *SHPFileGetInfo(ShapeBaseName\$, Entities, ShapeType)*

Parameters:

ShapeBaseName\$ The full path. No extension is needed, as the **SHP**, **SHX** and **DBF** extensions are added automatically.

Entities The number of objects in the shape file set. These are referenced from 0.

ShapeType The type of objects in the shape file set. Possible values are

- 1 = **POINT**,
- 3 = **ARC**,
- 5 = **POLYGON**,
- 8 = **MULTIPOINT**,
- 11 = **POINTZ**,
- 13 = **ARCZ**,
- 15 = **POLYGONZ**,
- 18 = **MULTIPOINTZ**,
- 21 = **POINTM**,
- 23 = **ARCM**,
- 25 = **POLYGONM**,
- 28 = **MULTIPOINTM**
- 31 = **MULTIPATCH**

Remarks:

If the file cannot be found or opened, Entities and ShapeType return 0.

Related Commands:

[SHPFileCreate](#) [SHPFileGetInfo](#) [SHPWriteAttribute](#)

SHPGetAttributeCount

geoPIXCL command. Reads the specified SHP/SHX/DBF file, and returns the number of attributes that are present.

Syntax: `SHPGetAttributeCount(ShapeBaseName,$, Count)`

Parameters:

ShapeBaseName\$ The full path. No extension is needed, as the **SHP**, **SHX** and **DBF** extensions are automatically added as required.

Count The number of attributed located in the file.

Related Commands:

[SHPRReadAttribute](#) [SHPWriteAttribute](#)

SHPGetFieldInfo

geoPIXCL command. Reads the specified **DBF** file, and returns the details of a field.

Syntax: *SHPGetFieldInfo(ShapeBaseName\$, FieldNumber, FieldName\$, Type, Width, DecimalPlaces)*

Parameters:

| | |
|------------------------|--|
| <i>ShapeBaseName\$</i> | The full path. No extension is needed, as the SHP , SHX and DBF extensions are automatically added as required. |
| <i>FieldNumber</i> | The 0-indexed field to read. |
| <i>FieldName\$</i> | The name of the field, read from the file. |
| <i>Width</i> | The width of the field in bytes. |
| <i>DecimalPlaces</i> | The number of decimal places for DOUBLE data. Otherwise 0. |

Related Commands:

[SHPReadAttribute](#) [SHPWriteAttribute](#)

SHPReadAttribute

geoPIXCL command. Reads the specified attribute value from the **DBF** file in the Shape file set.

Syntax: `SHPReadAttribute(ShapeBaseName$, Record, Field, Value|${&, Result)`

Parameters:

| | |
|------------------------|---|
| <i>ShapeBaseName\$</i> | The full path. No extension is needed, as the SHP , SHX and DBF extensions are automatically added as required. |
| <i>Record, Field</i> | The record and field numbers to read. |
| <i>Value \${&</i> | The appropriate type for the requested field. Note that setting a string variable and asking for an INTEGER or DOUBLE type field will return the string representation of the number. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[SHPAddAttribute](#) [SHPWriteAttribute](#)

SHPReadObject

geoPIXCL command. Reads the specified SHP/SHX file, and returns the information of the selected entity i.e. a particular shape within a shape file. The files are opened, read and closed by this command.

The arrays must be at least as large as the number of elements in the entity. Use the [SHPPFileGetInfo](#) command to find the necessary size. Note that each entity will be of the same type, but does not have to have the same number of vertices.

Syntax: *SHPPReadObject(ShapeBaseName\$, Entity, ShapeID, Parts, StartPart, X& [StartIndex] , Y&[StartIndex], Z&[StartIndex], M&[StartIndex] , Bounds&[StartIndex], Return)*

Parameters:

| | |
|-----------------------------------|--|
| <i>ShapeBaseName\$</i> | The full path. No extension is needed, as the SHP , SHX and DBF extensions are automatically added as required. |
| <i>Entity</i> | The entity to read. |
| <i>ShapeID</i> | The type of entity. One of the following POINT, ARC, POLYGON, MULTIPOINT, POINTZ, AR CZ, POLYGONZ, MULTIPOINTZ, POINTM, AR CM, POLYGONM, MULTIPOINTM, MULTIPATCH |
| <i>Parts</i> | The number of parts in the entity. |
| <i>StartPart</i> | The start ID |
| <i>X& [StartIndex]</i> | The X coordinate array, and start index for loading data. |
| <i>Y& [StartIndex]</i> | The Y coordinate array, and start index for loading data. |
| <i>Z& [StartIndex]</i> | The Z coordinate array, and start index for loading data. |
| <i>M& [StartIndex]</i> | The M coordinate array, and start index for loading data. |
| <i>Bounds& [StartIndex]</i> | The XYZM bounds array, and start index for loading data. |
| <i>Result</i> | 1 if the read was successful, otherwise 0. |

Related Commands:

[SHPPFileCreate](#) [SHPPCreateSimpleObject](#) [SHPPFileGetInfo](#) [SHPPReadAttribute](#)

SHPReadObjectVertices

geoPIXCL command. Reads the specified SHP/SHX file, and returns the information of the selected entity i.e. a particular shape within a shape file. The files are opened, read and closed by this command. Note that each entity will be of the same type, but does not have to have the same number of vertices.

Syntax: *SHPReadObjectVertices(ShapeBaseName\$, Entity, ShapeID, Parts, StartPart, Vertices)*

Parameters:

| | |
|------------------------|--|
| <i>ShapeBaseName\$</i> | The full path. No extension is needed, as the SHP, SHX and DBF extensions are automatically added as required. |
| <i>Entity</i> | The entity to read, numbered from 0. |
| <i>ShapeID</i> | The type of entity. One of the following POINT, ARC, POLYGON, MULTIPOINT, POINTZ, AR CZ, POLYGONZ, MULTIPOINTZ, POINTM, AR CM, POLYGONM, MULTIPOINTM, MULTIPATCH |
| <i>Parts</i> | The number of parts in the entity. |
| <i>StartPart</i> | The start ID |
| <i>Vertices</i> | The number of vertices in this object. This count can be used to set the size for the arrays required by a subsequent call to SHPReadObject . |

Related Commands:

[SHPFileCreate](#) [SHPCreateSimpleObject](#) [SHPFileGetInfo](#)

SHPWriteAttribute

geoPIXCL command. Writes the specified attribute value to the **DBF** file in the Shape file set.

Syntax: `SHPWriteAttribute(ShapeBaseName$, Record, Field, Value|$, &, Result)`

Parameters:

| | |
|------------------------|--|
| <i>ShapeBaseName\$</i> | The full path. No extension is needed, as the SHP , SHX and DBF extensions are automatically added as required. |
| <i>Record, Field</i> | The record and field numbers to read. |
| <i>Value \$, &</i> | The appropriate type for the requested field. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[SHPAddAttribute](#) [SHPReadAttribute](#)

Shutdown

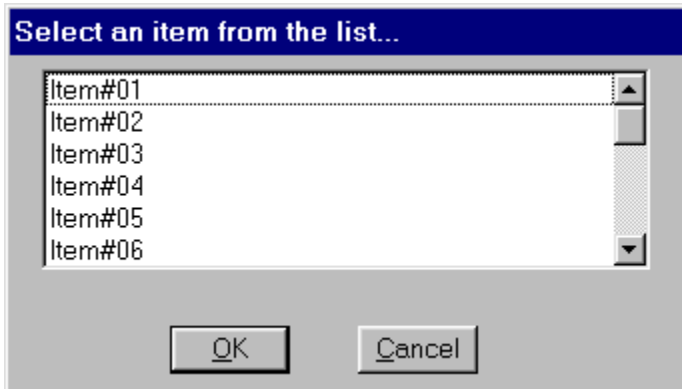
Windows NT / 2000 command only! Shuts down Windows as though you had clicked **Start: Shutdown**. It also displays a message box indicating the impending shutdown, and offers several options for controlling the shutdown process.

Syntax: `Shutdown(ComputerName$,ShutdownMsg$,Timeout,RESTART/NORESTART)`

Parameters:

- ComputerName\$** The network name of the computer you want to shut down. The name must be in the UNC form—for example, "\\VYSOR_P200". If you use a null string (" "), PiXCL shuts down the local computer.
- ShutdownMsg\$** The message to be displayed within the shutdown message box.
- Timeout** The number of seconds before shutdown takes place. Also, the number of seconds that the message box is displayed indicating the pending shutdown. If *Timeout* is set to zero, no message box is displayed.
- RESTART** Restarts the computer immediately after shutdown.
- NORESTART** Displays a restart message box, allowing you to manually control the restart process.

A dialog something like show below will appear.



Remark:

You cannot shut down a computer unless you have the proper rights (e.g. Administrator) to do so.

You can cancel the shutdown process by issuing an [AbortShutdown](#) command (see the example).

If this command is used in Windows 95 or 98, a message box appears to inform you that the command is not supported. The program will then continue.

Example:

This example shuts down the PC named VYSOR_P200, displaying a message box for 20 seconds before completing the shutdown process. If the user presses ESC while the message box is displayed, the shutdown is aborted using the [AbortShutdown](#) command.

```
{Shut down PC named VYSOR_P200,}
Shutdown("\\VYSOR_P200",
        "Click on the main window and press Esc to cancel shutdown",
        20,NORESTART)
```

```
{If the user mashes ESC, bail out of shutdown}  
Key = 27          {Virtual key for Esc}  
SetKeyboard(Key,Abort)
```

```
Wait_for_input:  
    WaitInput()
```

```
Abort:  
    AbortShutdown  
    Goto Wait_for_input
```

Related Commands:

[AbortShutdown](#), [Logoff](#), [ExitWindows](#)

Sin

Floating Point math library function. Calculate the Sine of an angle in radians.

Syntax: `Sin(Angle&, Value&)`

Parameters:

`Angle&` The angle in radians
`Value&` The result of the function.

Related Commands:

[Cos](#) [Tan](#)

Sinh

Floating Point math library function. Calculate the hyperbolic sine of an angle in radians.

Syntax: `Sinh(Angle&, Value&)`

Parameters:

Angle& The angle in radians
Value& The result of the function.

Related Commands:

[Cosh](#) [Tanh](#)

Space

Initializes a string variable to a specified number of spaces.

Syntax: `Space(String$,Length)`

Parameters:

`String$` The string variable you want to initialize.

`Length` The number of spaces you want to place in `String$`.

Remarks:

The Space command overwrites the existing contents of `String$`.

If you want to add spaces to the end of a string, use the Pad command. If you want to add spaces to the start of a string, use the + operator to concatenate two strings together, as in `Variable$ = " " + "Ottawa"`.

Example:

The following example initializes the Title\$ variable to 40 spaces. It then appends the text "A Not So Centered Title" to the end of Title\$ and draws the result on the screen.

```
Space(Title$,40)
Title$=Title$+"A Not So Centered Title"
DrawText(1,1,Title$)
WaitInput()
```

Related Commands:

[Pad](#), [Trim](#), [Set](#)

Sqrt

Floating Point math library function. Calculate the square root function, `sqrt(x)`.

Syntax: `Sqrt(Number&, Root&)`

Parameters:

`Number&` The positive number.

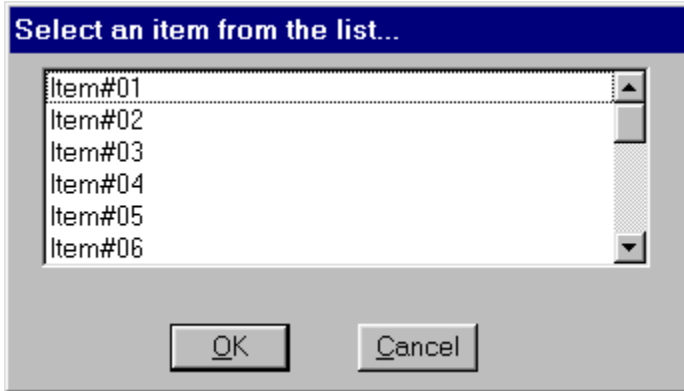
`Root&` The square root. If `Number&` is negative, `Root&` returns 0.0.

Related Commands:

[Exp](#)

StatusWindow

Enables or disables a standard status bar display window on your application window, either at the default position at the bottom of the client area, or optionally at the top of the client area, just below the menu bar. Status bars can also be split into up to four parts of varying lengths using the [DrawStatusWinText](#) command.



Syntax: [StatusWindow](#)([TOKEN_1](#), [TOKEN_2](#), [Parts](#),
[End#1](#),[End#2](#),[End#3](#),[End#4](#))

Parameters:

- [TOKEN_1](#) [ENABLE](#) makes the status bar visible, and [DISABLE](#) removes it. If [DISABLE](#) is specified, the other arguments are ignored.
- [TOKEN_2](#) [BOTTOM](#): Draw the status bar at the bottom of the client area. This is the most common position for Windows 95 and NT applications.
- [TOP](#): Draw the status bar at the top of the client area. Rarely used, as toolbars are generally placed at the top of the client area.
- [Parts](#) The number of sunken parts or sub-sections to be drawn, to a maximum of 4. If this number is zero, no sunken parts are drawn.
- [End#1 - End#4](#) The end pixel client co-ordinate of each part. A small gap is written between each part. If [End#n](#) is -1, the nth part is written to the right hand side of the client area. If [Parts](#) is zero, these values are ignored.

Remarks:

Status bars appear in gray, or to whatever color the system default has been set. Status bars will be redrawn automatically to the correct position if you resize the application window. Any text in the status bar will remain unchanged.

Related Command:

[DrawStatusWinText](#), [DrawStatusText](#), [ToolBar](#) , [ToolWindow](#)

StillImageAdmin

PIXCL 5 command. Windows 98 introduced the Still Image device concept and the Still Image Monitor (Stimon) that keeps track of devices that support Still Image aware applications. The general idea is that a device e.g. a scanner or digicam can initiate an image acquisition event, like pressing a button on the device, and the appropriate Still Image Registered application will be started. In Windows ME, this was extended to become Windows Image Acquisition, or WIA.

Syntax: `StillImageAdmin(INIT|INFO|REG|UNREG,AppName$,EventName$,Result)`

Parameters:

| | |
|--------------------------|---|
| <code>INIT</code> | Initializes the Still Image system, and makes the application Still Image aware. This is automatically undone when the PiXCL application exits. |
| <code>INFO</code> | Used to check if the application was started by a Still Image event. |
| <code>REG</code> | Register the Application and the EventName. |
| <code>UNREG</code> | Unregister the Application and the EventName. |
| <code>AppName\$</code> | The name of the application to be (un)registered, or the application that launched it. See Remarks below. |
| <code>EventName\$</code> | The name of the event to be (un)registered, or the event that launched the application. See Remarks below. |
| <code>Result</code> | 1 if the operation was successful, otherwise 0. |

Remarks:

When an application is registered with Still Image, the details get written to the Registry, and removed on an unregister operation. The Registry key in **HKEY_LOCAL_MACHINE** is

Software\Microsoft\Windows\CurrentVersion\StillImage\Registered Applications

`AppName$` becomes the key name e.g. `MyImagingApp`, and `EventName$` becomes (say)

`C:\PiXCLTools\Samples\MyImagingApp.EXE`. The registration process adds

`/StiDevice:%1 /StiEvent:%2` to `EventName$`.

When an application has been started by a Still Image event, a call to `StillImageAdmin` with the `INFO` token returns `EventName$` with a GUID, a number in the form “{6bded1fc5-810f-11d0-bec7-08102be2092f}”, otherwise `AppName$` and `EventName$` are empty strings.

Another way to check if the application was started by a Still Image event is to check the command line with the `GetCmdLine` function for a “`/StiDevice:<name> /StiEvent:<name>`” string.

Related Commands:

[GetCmdLine](#)

Str

Converts an integer to a string.

Syntax: *Str(Number,String\$)*

Parameters:

Number The integer you want to convert; it must be in the range -2147483647 to 2147483648.

String\$ The string variable that will contain the result.

Example:

The following program creates a purple brush and draws a rectangle using it. It then converts the Red, Green, and Blue integers to strings. From these results, the program builds the string "The RGB value is 128,0,63" and places it on the screen below the rectangle.

```
Red=128
Green=0
Blue=63
UseBrush (SOLID,Red,Green,Blue)
DrawRectangle (10,10,30,20)
Str (Red,Str1$)
Str (Green,Str2$)
Str (Blue,Str3$)
Out$ = "The RGB value is "+Str1$
Out$ = Out$+", "
Out$ = Out$+Str2$
Out$ = Out$+", "
Out$ = Out$+Str3$
DrawText (10,25,Out$)
WaitInput()
```

Related Command:

[Val](#)

Str64

PIXCL 5.1 command. Converts a 64-bit integer to a string.

Syntax: *Str64(Number64#,String\$)*

Parameters:

Number64# The integer you want to convert.
String\$ The string variable that will contain the result.

Related Command:

[Val64](#)

StrCmp

Performs a case-sensitive comparison of two strings and returns an integer value indicating their relationship.

Syntax: `StrCmp(String1$,String2$,Result)`

Parameters:

String1\$,String2\$ The strings you want to compare.

Result An integer variable that will contain one of the following values based on the result of the comparison:

| | |
|---|-----------------------|
| 0 | String1\$ < String2\$ |
| 1 | String1\$ = String2\$ |
| 2 | String1\$ > String2\$ |

Remark:

The comparison is made based on the current language (set in Control Panel).

Example:

The following program asks you to enter two strings. It then compares them and displays a message box indicating their relationship. For example, if you enter "Zaph" for the first string and "Dingbats" for the second, the message box will display "Zaph is greater than Dingbats."

```
TextBox("Enter one string","",String1$,Button)
TextBox("Enter another string","",String2$,Button)
StrCmp(String1$,String2$,Rel)
If Rel = 0 Then Rel$ = " is less than " | Goto Build_Message
If Rel = 1 Then Rel$ = " is equal to " | Goto Build_Message
Rel$ = " is greater than "
Build_Message:
Out$ = String1$ + Rel$
Out$ = Out$ + String2$
MessageBox(OK,1,INFORMATION,Out$,"Results of StrCmp",Temp)
```

Related Command:

[StrCmpi](#)

StrCmpl

Performs a case-insensitive comparison of two strings and returns an integer value indicating their relationship.

Syntax: `StrCmpl(String1$,String2$,Result)`

Parameters:

`String1$,String2$` The strings you want to compare.

Result An integer variable that will contain one of the following values based on the result of the comparison:

- 0 String1\$ < String2\$
- 1 String1\$ = String2\$
- 2 String1\$ > String2\$

Remark:

The comparison is made based on the current language (set in Control Panel).

Example:

The following program is a variation of the one shown for StrCmp. It compares two strings and displays a message box indicating their relationship.

```
TextBox("Enter one string","",String1$,Button)
TextBox("Enter another string","",String2$,Button)
StrCmpI(String1$,String2$,Rel)
If Rel = 0 Then Rel$ = " is less than " | Goto Build_Message
If Rel = 1 Then Rel$ = " is equal to " | Goto Build_Message
Rel$ = " is greater than "
Build_Message:
Out$ = String1$ + Rel$
Out$ = Out$ + String2$
MessageBox(OK,1,INFORMATION,Out$,"Results of StrCmpI",Temp)
```

Related Command:

[StrCmp](#)

StrRepl

Replaces one string with another in a target string. This can be handy when you need to replace, for example, a filename extension with a different extension.

Syntax: *StrRepl*(*TargetString\$,OldString\$,NewString\$,Result*)

Parameters:

TargetString\$

The string that contains the substring you want to replace. This can be a NULL string, in which case the result is *NewString\$*.

OldString\$

The string variable that contains the substring to be replaced.

NewString\$

The string variable that contains the substring that is to replace the old string. This can be a NULL string if required.

Result

If the operation was successful, *Result* returns a value of 1, otherwise it returns zero.

Remarks:

Result will return zero if the *OldString\$* does not exist in the *TargetString\$*.

Related Command:

All the string commands.

StrReplAll

Replaces all instances of one string with another in a target string. This can be achieved with a For-Next loop, but is quicker for large strings with many instances of the string to be replaced.

Syntax: *StrReplAll(TargetString\$,OldString\$,NewString\$,Result)*

Parameters:

TargetString\$

The string that contains the substring you want to replace. This can be a NULL string, in which case the result is *NewString\$*.

OldString\$

The string variable that contains the substring to be replaced.

NewString\$

The string variable that contains the substring that is to replace the old string. This can be a NULL string if required. The effect is removing the *OldString\$*.

Result

If the operation was successful, *Result* returns a value of 1, otherwise it returns zero.

Remarks:

Result will return zero if the *OldString\$* does not exist in the *TargetString\$*.

Related Command:

All the string commands.

StrRev

Reverses the character order of a string.

Syntax: `StrRev(TargetString)`

Parameters:

TargetString The string that you want to reverse.

Related Command:

All the string commands.

Substr

Extracts one string (a *substring*) from another.

Syntax: `Substr(String$,Places,Location,Result$)`

Parameters:

| | |
|-----------------------|--|
| <code>String\$</code> | The string from which you want to extract a substring. |
| <code>Places</code> | The number of characters you want to extract. (If the number you specify is greater than the number of characters remaining in <code>String\$</code> , PiXCL extracts up to the end of <code>String\$</code> ; see the example.) |
| <code>Location</code> | A number indicating where in <code>String\$</code> to start extracting. (The first position in <code>String\$</code> is 1.) |
| <code>Result\$</code> | A string variable that will contain the result. |

Remark:

Starting at `Location`, this function extracts from `String$` the number of characters indicated by `Places` and stores the resulting substring in `Result$`.

Example:

This program uses the `Instr` function to determine the location of the first blank space in the string "Hello World!" It then extracts the remainder of the string following the space and places the result on the screen.

```
Text$ = "Hello World!"
Instr(Text$, " ", Location)
Location = Location + 1
Substr(Text$, 99, Location, Result$)
DrawText(10, 10, Result$)
WaitInput()
```

Related Commands:

[Instr](#), [Left](#), [Right](#), [Len](#)

Switch

PIXCL 5 command. An extension of the [If](#) statement, the [Switch](#) command provides a means to check any number of possible integer values, and handle them appropriately.

Syntax: [Switch](#) ([Integer_Variable](#))
[Case](#) ([Integer_value](#))
...[commands](#)
[Break](#)
[Default](#)
... [commands](#)
[EndSwitch](#)

Remarks:

Each case has to be handled separately. Sequential Case statements are not supported (yet). The [Break](#) Keyword is required if the case value functions are complete, and program execution jumps to the commands following the [EndSwitch](#) keyword. Switch structures can be embedded within each other if desired.

Placing a [Goto](#) statement inside a [Switch](#) structure is not supported, and will eventually lead to a syntax error in code that previously worked.

Related Commands:

[If-Then](#) [If-Else-Endif](#)

SysCmdEndAt

There are times when you will want to save some of the work or data that has been done, and this is normally done with a menu option e.g. **File->Exit**. The exit label processing will save the current work or data.

The Close Window button on a main window has the default operation of closing the PIXCL application immediately, resulting in possible lost data. The [SysCmdEndAt](#) command provides a jump to label.

Syntax: [SysCmdEndAt\(Label\)](#)

Parameter:

[Label](#) The jump-to label name. If [Label](#) does not exist in the script, a syntax error is generated.

Related Commands:

[PXLResumeAt](#)

Tan

Floating Point math library function. Calculate the tangent of an angle in radians.

Syntax: `Tan(Angle&, Value&)`

Parameters:

Angle& The angle in radians
Value& The result of the function.

Related Commands:

[Sin](#) [Cos](#)

Tanh

Floating Point math library function. Calculate the hyperbolic tangent of an angle in radians.

Syntax: `Tanh(Angle&, Value&)`

Parameters:

`Angle&` The angle in radians
`Value&` The result of the function.

Related Commands:

[Sinh](#) [Cosh](#)

TaskBarIcon

The Windows 95 and NT 4.0 TaskBar includes a notification area where an application can put an icon to indicate the status of an operation or to notify the user about an event. For example, an application might put a printer icon in the TaskBar to show that a print job is under way. The notification area is at the right end of the TaskBar (if the TaskBar has a horizontal orientation) or at the bottom (if the TaskBar has a vertical orientation).

An icon in the TaskBar can have a tooltip associated with it, and this appears if the mouse is moved over the icon in the notification area.

Syntax: `TaskBarIcon(DELETE|ADD|MODIFY,IconNumber,ICONid,
ToolTipText$,Result)`

Parameters

| | |
|----------------------|--|
| <code>DELETE</code> | Remove the specified icon. |
| <code>ADD</code> | Add an icon with the specified tool tip. |
| <code>MODIFY</code> | Modify an existing Task Bar icon, function or tool tip. |
| <i>IconNumber</i> | An application defined number. Can be positive or negative. Best used as a numeric variable. |
| <code>ICONid</code> | One of <code>ICON01</code> - <code>ICON16</code> , <code>QUESTION</code> , <code>ASTERISK</code> , <code>EXCLAMATION</code> , <code>STOP</code> . i.e. same as the DrawIcon command. |
| <i>ToolTipText\$</i> | An icon in the TaskBar can have a tooltip control associated with it. The text string is limited to a maximum of 64 characters. If the string is <code>NULL ""</code> , no tool tip will be displayed. |
| <i>Result</i> | 1 if the operation succeeds, otherwise 0. |

Remarks

If you close the PiXCL application that wrote the icon into the TaskBar, Windows will remove it within a few seconds. Good programming practice however suggests that you should clean up before you end a program.

If an `IconNumber` does not exist, *Result* returns 0.

If you left click on the mouse, interpreting will resume from the current position i.e. the [WaitInput\(\)](#).

If you right click, the pixel application is sent a right click message, that is, the equivalent of right clicking in a defined region in the PiXCL application client area. If a [SetRightMouse](#) command is valid, the application will process according to the code.

If you double left click, the pixel application is sent a double left click message, that is, the equivalent of double left click clicking in a defined region in the PiXCL application client area. If a [SetDbfMouse](#) command is valid, the application will process according to the code.

Example

This simple example shows how an icon can be written to the notification area, and the window minimized. The PiXCL application will continue to process in the background. In this example, it sits in a [WaitInput\(\)](#) loop. If you move the mouse over the icon, the specified tool tip will be displayed.

If you left click on the mouse, interpreting will resume from the current position (i.e. the [WaitInput\(\)](#)), and the window will be

restored.

Initialize:

```
WinGetActive(Win$)
UseCoordinates(PIXEL)
WinLocate(Win$,100,100,600,450,Res)
```

```
Title$ = "Task Bar Icon Test"
WinTitle(Win$, Title$)
InfoMenu(REMOVE)
WaitInput(100)
SetMenu("&File", IGNORE,
    "TaskBarAdd 1", TaskBarAdd1,
    "TaskBarDel 1", TaskBarDell,
    "TaskBarMod 1", TaskBarMod1,
    SEPARATOR,
    "Exit!", Leave,
    ENDPOPUP)
```

Wait_for_Input:

```
WaitInput()
```

TaskBarAdd1:

```
TaskBarIcon(ADD,1,ICON01,"PiXCL Test 1",Res)
WinShow(Title$,MINIMIZE,Res)
WaitInput()
```

```
WinShow(Title$,RESTORE,Res)
Goto Wait_for_Input
```

TaskBarMod1:

```
TaskBarIcon(MODIFY,1,ICON10,"PiXCL Test 2a",Res)
WaitInput(500)
TaskBarIcon(MODIFY,1,ICON11,"PiXCL Test 2b",Res)
WaitInput(500)
TaskBarIcon(MODIFY,1,ICON12,"PiXCL Test 2c",Res)
WaitInput(500)
TaskBarIcon(MODIFY,1,ICON13,"",Res)
Goto Wait_for_Input
```

TaskBarDell:

```
TaskBarIcon(DELETE,1,ICON01,"PiXCL Test",Res)
Goto Wait_for_Input
```

Leave:

```
End
```

TextBox

Puts up a dialog box with a single-line edit control. You can use it to get one line of input from the user.

Syntax: `TextBox(Text$,Caption$,Input$,ButtonPushed)`

Parameters:

Text\$ The message to be displayed within the dialog box.

Caption\$ The text you want to appear in the title bar of the dialog box.

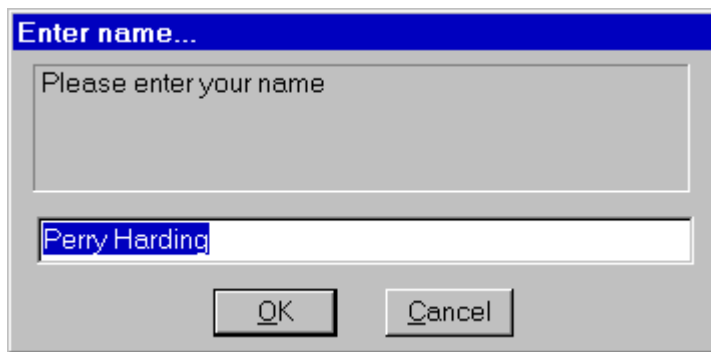
Input\$ A string variable that will contain the text that is entered. If you choose Cancel (or press ESC) to leave the dialog box, *Input\$* is assigned a null string ("").

ButtonPushed An integer variable that returns a number corresponding to the button that was pushed to leave the dialog box. OK is assigned the number 1, and Cancel is assigned the number 2.

Remark:

By initializing *Input\$*, you can have some default text appear in the edit control (see the example).

Example:



This example displays a text box requesting you to enter your name, as shown at left. After you enter your name and select OK, the program displays the result in a message box.

TextBox solicits a single line of input.

```
Text$="Please enter your name"  
Caption$="Enter name..."  
Name$="Perry Harding"  
TextBox(Text$,Caption$,Name$,Pushed)  
If Pushed = 2 Then End {Cancel selected, so quit}  
Out$ = "Your name is " + Name$  
Message:  
MessageBox(OK,1,NOICON,Out$,"Name",Temp)
```

Related Commands

[ListBox](#), [MessageBox](#)

TextBoxExt

This command extends the text box dialog with a large text area and Help button that displays context sensitive help in a standard MessageBox.

Syntax: `TextBoxExt(Text$,Label$,Help$,Input$,Btn)`

Parameters:

| | |
|----------------|--|
| <i>Text\$</i> | The static text in the dialog |
| <i>Label\$</i> | The text in the title bar |
| <i>Help\$</i> | The text in the Help MessageBox. |
| <i>Input\$</i> | String displayed in the edit control. |
| <i>Btn</i> | The button pushed. OK = 1, CANCEL = 2, Help = 3. |

Example:

```
ExtdText:
    Text$ = "Extended text box text field"
    Label$ = "Enter some data from the keyboard"
    Input$ = "blah blah blah"
    Help$ = "This should be context sensitive help"
    TextBoxExt(Text$,Label$,Help$,Input$,Btn)
    Goto Wait_for_Input
```

Related Commands:

[TextBox](#), [MessageBox](#)

TileBitmapWindows

PIXCL 5 command. Up to eight bitmap windows can be created within a PiXCL client area. Where you need to have multiple images displayed over the whole client area, the bitmap windows can be tiled in vertical or horizontal mode.

Syntax: [TileBitmapWindows\(VERT|HORZ,Result\)](#)

Parameters:

[VERT](#) Bitmap windows are maximized in the current PiXCL client area, vertically across the screen.
[HORZ](#) Bitmap windows are maximized in the current PiXCL client area, horizontally across the screen.
[Result](#) The number of bitmap windows that have been tiled. If no bitmap windows are present, Result returns 0.

Remarks:

The tiling process takes in to account the presence or absence of the toolbar and status bar, and adjusts the position accordingly. Hence, if you have tiled bitmap windows, and you resize the PiXCL client area, you would need to issue the [TileBitmapWindows](#) command again in the processing of a [WinResizeAt](#) command.

Related Commands:

[DrawBitmapWindow](#) [WinResizeAt](#)

TimeToASCII

The current system or local time can be returned in a string that can be used directly in applications or other commands, such as Drawtext.

Syntax: `TimeToASCII(SYSTEM | LOCAL, mode_TOKEN, TimeString$)`

Parameters:

| | |
|-----------------------------|--|
| <code>SYSTEM LOCAL</code> | Defines the system or current time to be returned. |
| <code>mode_TOKEN</code> | All example dates below are the same. |
| <code>MMDDYYYY</code> | e.g. 2/8/1997 |
| <code>DDMMYYYY</code> | e.g. 8/2/1997 |
| <code>WDDMYYYY</code> | e.g. Saturday, 8 February, 1997 |
| <code>WDMDDYYYY</code> | e.g. Saturday, February 8, 1997 |
| <code>MDDYYYY</code> | e.g. February 8, 1997 |
| <code>TimeString\$</code> | The returned string in the above format. |

Related Commands:

[GetLocalTime](#), [GetSystemTime](#), [SetLocalTime](#), [SetSystemTime](#), [GetTimeZone](#)

Toolbar

PIXCL 4.1 and later supports a standard Windows type application toolbar to simplify menu operations, or to provide button controls when using a menu is the second choice, such as selection of draw and paint tools. In PiXCL, only one toolbar at the top of the screen under the menubar is supported, and it can have a maximum of 64 buttons. Tooltips are supported in the command, and are automatically displayed in the standard Windows fashion.

The `Toolbar` command has a variable number of arguments, in the same fashion as the `SetMenu`, `SetPopupMenu`, the `SetMouse` commands, `SetEditControl`, and `ComboBox`.

A Toolbar can be either RAISED mode (i.e. 3D), or the newer FLAT mode that appeared with Microsoft's Internet Explorer and in Windows 98 / 2000.

Syntax: `Toolbar()` to remove the toolbar
`Toolbar(mode_TOKEN, size_TOKEN,
button_TOKEN, state_TOKEN, style_TOKEN, ToolTip$, label,. . .)`

Parameters:

| | | | |
|-------------------------|--------------------------------------|------------------|--------------------------|
| <code>mode_TOKEN</code> | <code>RAISED FLAT</code> | must be present. | |
| <code>size_TOKEN</code> | <code>STD_SMALL STD_LARGE</code> | | from common controls DLL |
| | <code>VIEW_SMALL VIEW_LARGE</code> | | from common controls DLL |
| | <code>HIST_SMALL HIST_LARGE</code> | | from common controls DLL |
| | <code>PXL_SMALL PXL_LARGE</code> | | built into PiXCL. |

then sets of the following ...

| | |
|---------------------------|---|
| <code>button_TOKEN</code> | button bitmap, see below. |
| <code>state_TOKEN</code> | <code>ENABLED CHECKED DISABLED NULL</code> |
| <code>style_TOKEN</code> | <code>STD CHECK STD_G CHECK_G SEPARATOR</code> |
| <code>ToolTip\$</code> | Up to 79 characters, truncated if longer, can be null string. |
| <code>label</code> | Jump to label. Will often be same as one of your menu labels. |

Toolbar Button Bitmap Tokens

Toolbars are created using the Windows Common Controls DLL, which includes three sets of toolbar buttons for the so-called application frame or Standard operations, application View operations, and History operations.

PIXCL includes the above buttons built-in, and also provides an additional set of built-in buttons bitmaps designed for typical paint, drawing tool select and general purpose buttons. Standard, View and History TOKEN bitmaps built into PiXCL have the prefix `PXL_`. The following TOKEN values are used as indexes to the Standard, View and History bitmaps.



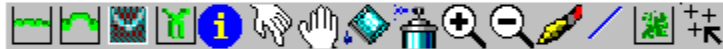
Standard ToolBar buttons in L-R order.



View ToolBar buttons in L-R order.



History ToolBar buttons in L-R order.



PIXCL 4 Toolbar buttons in L-R order

Additional PIXCL 5 buttons

Additional geoPIXCL Toolbar buttons in L-R order.



Standard Bitmap TOKENS in Order, as shown above.

| TOKEN name | Typical use |
|----------------------------|---|
| CUT, PXL_CUT | Edit operations: Cut out to clipboard |
| COPY, PXL_COPY | Edit operations: Copy to clipboard |
| PASTE, PXL_PASTE | Edit operations: paste from clipboard |
| UNDO, PXL_UNDO | Undo the last edit operation. |
| REDO, PXL_REDO | Redo the last edit operation. |
| DELETE, PXL_DELETE | Delete the selected object. |
| FILENEW, PXL_FILENEW | Create a new file of a specified type. |
| FILEOPEN, PXL_FILEOPEN | Open an existing file of a specified type. This button would generally invokes a FileGet command. |
| FILESAVE, PXL_FILES SAVE | Save the current file. A FileSave command would generally be invoked. |
| PROPERTIES, PXL_PROPERTIES | Display properties of an object, perhaps in a dialog box or as text and images in the client area. |
| PRINTPRE, PXL_PRINTPRE | Display a print preview. This could be by running an external program or Windows utility. |
| SHOHELP, PXL_SHOHELP | Invoke some sort of Help function. A WinHelp or WinHTMLHelp command can be used here. |
| FIND, PXL_FIND | Find the next occurrence of a string or object. |
| REPLACE, PXL_REPLACE | Replace the selected object with another object of the same or different type. |
| PRINT, PXL_PRINT | Print a current document. This may involve running an external program or Windows utility. |

View Bitmap TOKENS in Order, as shown above.

| TOKEN name | Typical use |
|----------------------------|--|
| LARGEICONS, PXL_LARGEICONS | Display a file tree view with large icons. |

| | |
|-----------------------------------|---|
| SMALLICONS, PXL_SMALLICONS | Display a file tree view with small icons. |
| VIEW_LIST, PXL_VIEWLIST | View a List of objects, typically filenames, but can be anything. |
| DETAILS, PXL_DETAILS | Create a dialog or other display showing more details about an object. |
| SORTBYNM, PXL_SORTBYNM | Sort a list by name. |
| SORTBYSZ, PXL_SORTBYSZ | Sort a list by file size. |
| SORTBYDT, PXL_SORTBYDT | Sort a list by date. |
| SORTBYTYP, PXL_SORTBYTYP | Sort a list by data or file type. |
| PARENTFOLDER, PXL_PARENTFOLDER | Jump back to the parent folder of a file. |
| NETCNCT, PXL_NETCNCT | Connect to a network, if available. This will require running external programs or Windows utilities. |
| NETDSCNCT, PXL_NETDSCNCT | Disconnect to a network. This will require running external programs or Windows utilities. |
| NEWFOLDER, PXL_NEWFOLDER | Create a new folder. A DirMake command would be used for this. |

History Bitmap TOKENs in Order, as shown above.

| TOKEN name | Typical use |
|---------------------------------------|--|
| BACK, PXL_BACK | Jump back to the previous display window history list. Used in Web browsers typically. |
| FORWARD, PXL_FORWARD | Jump forward to the next display window history list. Used in Web browsers typically. |
| FAVORITES, PXL_FAVORITES | Display a list of recently accessed locations. This might be in an INI type file. See FileRead_INI . |
| ADDTOFAVORITES, PXL_ADDTOFAVORITES | Add to a list of recently accessed locations. This might be in an INI type file. See FileWrite_INI . |
| VIEWTREE, PXL_VIEWTREE | View a list in a tree structure display. |

NULL Only used with [SEPARATOR](#) keyword.

PIXCL Bitmap TOKENs in Order, as shown above.

| TOKEN name | Typical use |
|------------|--|
| ANNOT | Invoke dialogs to annotate an image. |
| SQSELECT | Select a square region with mouse. |
| POLYSELECT | Select a polygon shaped region with the mouse. |
| ERASE | Erase pixels in an image or in the client area. |
| OPENIMAGE | Open an image file. |
| HISTOGRAM | Create a histogram of an image or image set. |
| MIRROR | Perform a horizontal mirror operation on an image. |

| | |
|-----------|---|
| FLIP | Flip an image vertically. |
| ROTATE | Rotate an image an arbitrary number of degrees. |
| SHOWRGB | Show the RGB values of a pixel, or some other operation using RGB values. |
| PICKRGB | Use the mouse to select a pixel to report RGB colors. |
| CLASSIFY | Perform a classification of an image according to pixel color. |
| NORMALIZE | Process an image to normalize the RGB histograms. |
| EQUALIZE | Process an image to equalize the RGB histograms. |
| NEGATIVE | Invert the colors of an image, creating a color negative effect. |
| GAMMACRCT | Process an image to correct for gamma. |
| IMAGEINFO | Display image information. This might be in an ImageBox or MessageBox . |
| FINGER | Use the mouse to point to or select some object. |
| HAND | Grab an object and drag it with the mouse. |
| FLOOD | Perform a flood fill operation. |
| SPRAY | Perform a color spray paint operation. |
| ZOOMUP | Zoom an image up for more detail. |
| ZOOMDN | Zoom an image down to get a synoptic view. |
| BRUSH | Select a brush. You could use ChooseColor and UseBrush commands. |
| PEN | Select a pen. You could use ChooseColor and UsePen commands. |
| SCATTRGRM | Create and display a scattergram of two images. This requires third part utilities. |
| DIGITIZE | Digitize points from the client area, or from an external device or application. |
| CDROM | Perfrom some operation using a CD-ROM. |
| SAVEALL | Save all the current open files. |
| HELPER#1 | Start a Helper Application #1. |
| HELPER#2 | Start a Helper Application #2. |
| HELPER#3 | Start a Helper Application #3. |
| HELPER#4 | Start a Helper Application #4. |
| HELPINFO | Get on-line help information. |

Extra PiXCL 5 buttons.

Also available in geoPiXCL.

| | |
|--------------|---|
| DIALOG | Display a dialog. |
| RECYCLE | Perform a Recycle bin operation. |
| CAMERA | Perform a digital camera operation. |
| SCANNER_1 | Perform a scanner operation. |
| GPS | Do something related to a Global Positioning System data source or set. |
| ERASE_2 | Another Erase button. |
| HELPCONTEXT | Do a Help Context operation. |
| SCANNER_2 | Perform a scanner operation. |
| SELECTSOURCE | Display the TWAIN Select Source dialog. |
| SCANNER_3 | Perform a scanner operation. |
| MEASURE | Measure an image. |

| | |
|-------------------------------|--|
| TOGGLE | Toggle some operation. |
| GeoPiXCL buttons only. | <u>These bitmaps are not available in PiXCL44 or PiXCL50.</u> |
| POINTS | Digitize points. |
| LINES | Digitize lines. |
| POLYGONS | Digitize polygons. |
| LABELS | Digitize points for text labels. |
| NDVI | Normalized Difference Vegetation Index |
| AVGIMGSET | Average Image Set |
| SIGNATURE | Create a spectral signature file. |
| TRGAREAS | Define or extract training areas. |
| MLHCLAS | Maximum likelihood classifier. |
| PPDCLAS | Parallelipiped classifier. |
| PCAENHANCE | Principal components. |
| MARTAYMAP | Martin-Taylor inverse colour mapping. |
| DECORREL | Decorrelation stretch. |
| GLOBE | Global or geoid operation. |
| GEOTAG | Operations on geoTAGs in geoTIF files. |
| SHARPEN | Sharpen an image. |
| BROVEY | Brovey Transform Enhancement. |

Toolbar Button States

| | |
|----------|---|
| ENABLED | Button is functional. Most commonly used initial state. |
| CHECKED | Button is grayed and appears to be checked. |
| PRESSED | Button is grayed and appears to be pressed. |
| DISABLED | Button is grayed and non functional. |
| HIDDEN | Button is hidden. |
| NULL | Only used with SEPARATOR . |

Toolbar Button Styles

A button's style determines how the button appears and how it responds to user input. The [STD](#) style creates a toolbar button that behaves like a standard push button that pops out when you release it. A button that has the [CHECK](#) style is similar to a standard push button, except it toggles between the pressed and nonpressed states each time the user clicks it.

| | |
|---------------------------|--|
| STD | Draws the standard button that pops in and out when pressed. |
| CHECK | Draws the button in a checked state. |
| STD_G | Groups buttons. Groups are separated by any non-group button |
| CHECK_G | Draws a grouped button in checked state. |
| SEPARATOR | Draws a blank between buttons |

Grouping Buttons

You can create groups of toolbar buttons by using the [STD_G](#) or [CHECK_G](#) ([_G](#) means group the buttons) styles, causing a button to stay pressed until the user chooses another button in the group. The [SEPARATOR](#) style creates a small gap between buttons. A button with this style does not receive user input.

Toolbar Customization

By holding the SHIFT key, then selecting a button, you can either delete or move that button. All buttons can be moved or deleted.

To Delete a button: drag it into the client area and release the mouse.

To Move a button: drag it to another location in the toolbar and release the mouse. The button will be inserted to the left of the button at the release location. If you have defined two sequential [SEPARATOR](#) regions in the [Toolbar](#) command, you can insert a button between them.

Double Click on the toolbar background and the standard Windows Common Controls DLL toolbar customization dialog will be displayed.

Example:

See the sample program `toolbars.pxl`.

Related Commands:

[ChangeToolBarBtn](#) [ComboBox](#) [GetToolBarBtnStatus](#) [SetEditControl](#) [SetMenu](#) [SetPopupMenu](#), the [SetMouse](#) commands, [ToolWindow](#)

ToolWindow

This variable argument command is similar in syntax and operation as the [Toolbar](#) command, and lets you create any number of floating toolwindows anywhere in the screen i.e. popup toolwindows while a toolbar is created under the menubar of a PiXCL application.

The size and initial position are under your control, and individual toolwindows can overlap initially. ToolWindows have a smaller title bar than normal windows, and use a smaller font. ToolWindows are movable by selecting the titlebar and dragging with the mouse, but they cannot be resized.

Syntax:

```
ToolWindow()      to remove all toolwindows.
and
ToolWindow(x1,y1,x2,y2, POPUP, Title$,
           mode_TOKEN, size_TOKEN,
           button_TOKEN, state_TOKEN, style_TOKEN, ToolTip$, label, ...)
```

Parameters:

[x1,y1,x2,y2](#) The rectangle that defines the toolwindow. Note that the coordinates should be limited to the range of the current video screen i.e. regions between (0,0) (screen_Xmax, screen_Ymax). If the PiXCL application client area is smaller than the screen, which will usually be the case, you may create CHILD toolwindows that are not immediately visible. For the CHILD toolwindows [x1,y1,x2,y2](#) coordinates are client area coordinates, and for POPUP toolwindows, [x1,y1,x2,y2](#) are screen coordinates.

[POPUP](#) [POPUP](#) produces a toolwindow that floats anywhere on the screen. CHILD mode is not supported.

[Title\\$](#) The title string that appears on the toolwindow titlebar. It can be a null string "".

[mode_TOKEN](#) [RAISED](#) | [FLAT](#) must be present.
[size_TOKEN](#) [STD_SMALL](#) | [STD_LARGE](#) from common controls DLL
[VIEW_SMALL](#) | [VIEW_LARGE](#) from common controls DLL
[HIST_SMALL](#) | [HIST_LARGE](#) from common controls DLL
[PXL_SMALL](#) | [PXL_LARGE](#) built into PiXCL.

then sets of the following ...

[button_TOKEN](#) button bitmap, see below.
[state_TOKEN](#) [ENABLED](#) | [CHECKED](#) | [DISABLED](#) | [NULL](#)
[style_TOKEN](#) [STD](#) | [CHECK](#) | [STD_G](#) | [CHECK_G](#) | [SEPARATOR](#)
[ToolTip\\$](#) Up to 79 characters, truncated if longer, can be null string.
[label](#) Jump to label. Will often be same as one of your menu labels.

ToolWindow Button Bitmap Tokens

ToolWindows are created using the Windows Common Controls DLL, which includes three sets of toolbar buttons for the so-called application frame or Standard operations, application View operations, and History operations.

PiXCL includes the above buttons built-in, and also provides an additional set of built-in buttons bitmaps designed for typical paint, drawing tool select and general purpose buttons. Standard, View and History TOKEN bitmaps built into PiXCL have the

prefix `PXL_`. The following TOKEN values are used as indexes to the Standard, View and History bitmaps.



Standard ToolBar buttons in L-R order.



View ToolBar buttons in L-R order.



History ToolBar buttons in L-R order.



PIXCL 4 Toolbar buttons in L-R order

Additional PIXCL 5 buttons

Additional geoPIXCL Toolbar buttons in L-R order.



Standard Bitmap TOKENS in Order, as shown above.

| TOKEN name | Typical use |
|--|---|
| CUT, PXL_CUT | Edit operations: Cut out to clipboard |
| COPY, PXL_COPY | Edit operations: Copy to clipboard |
| PASTE, PXL_PASTE | Edit operations: paste from clipboard |
| UNDO, PXL_UNDO | Undo the last edit operation. |
| REDO, PXL_REDO | Redo the last edit operation. |
| DELETE, PXL_DELETE | Delete the selected object. |
| FILENEW, PXL_FILENEW | Create a new file of a specified type. |
| FILEOPEN, PXL_FILEOPEN | Open an existing file of a specified type. This button would generally invokes a FileGet command. |
| FILESAVE, PXL_FILESAVE | Save the current file. A FileSave command would generally be invoked. |
| PROPERTIES, PXL_PROPERTIES | Display properties of an object, perhaps in a dialog box or as text and images in the client area. |
| PRINTPRE, PXL_PRINTPRE | Display a print preview. This could be by running an external program or Windows utility. |
| SHOHLF, PXL_SHOHLF | Invoke some sort of Help function. A WinHelp or WinHTMLHelp command can be used here. |
| FIND, PXL_FIND | Find the next occurrence of a string or object. |
| REPLACE, PXL_REPLACE | Replace the selected object with another object of the same or different type. |
| PRINT, PXL_PRINT | Print a current document. This may involve running an external program or Windows utility. |

View Bitmap TOKENS in Order, as shown above.

| TOKEN name | Typical use |
|------------|-------------|
|------------|-------------|

| | |
|-----------------------------------|---|
| LARGEICONS, PXL_LARGEICONS | Display a file tree view with large icons. |
| SMALLICONS, PXL_SMALLICONS | Display a file tree view with small icons. |
| VIEW_LIST, PXL_VIEWLIST | View a List of objects, typically filenames, but can be anything. |
| DETAILS, PXL_DETAILS | Create a dialog or other display showing more details about an object. |
| SORTBYNM, PXL_SORTBYNM | Sort a list by name. |
| SORTBYSZ, PXL_SORTBYSZ | Sort a list by file size. |
| SORTBYDT, PXL_SORTBYDT | Sort a list by date. |
| SORTBYTYP, PXL_SORTBYTYP | Sort a list by data or file type. |
| PARENTFOLDER, PXL_PARENTFOLDER | Jump back to the parent folder of a file. |
| NETCNCNCT, PXL_NETCNCNCT | Connect to a network, if available. This will require running external programs or Windows utilities. |
| NETDSCNCT, PXL_NETDSCNCT | Disconnect to a network. This will require running external programs or Windows utilities. |
| NEWFOLDER, PXL_NEWFOLDER | Create a new folder. A DirMake command would be used for this. |

History Bitmap TOKENs in Order, as shown above.

| TOKEN name | Typical use |
|---------------------------------------|--|
| BACK, PXL_BACK | Jump back to the previous display window history list. Used in Web browsers typically. |
| FORWARD, PXL_FORWARD | Jump forward to the next display window history list. Used in Web browsers typically. |
| FAVORITES, PXL_FAVORITES | Display a list of recently accessed locations. This might be in an INI type file. See FileRead_INI . |
| ADDTOFAVORITES, PXL_ADDTOFAVORITES | Add to a list of recently accessed locations. This might be in an INI type file. See FileWrite_INI . |
| VIEWTREE, PXL_VIEWTREE | View a list in a tree structure display. |

NULL Only used with [SEPARATOR](#) keyword.

PiXCL Bitmap TOKENs in Order, as shown above.

| TOKEN name | Typical use |
|------------|---|
| ANNOT | Invoke dialogs to annotate an image. |
| SQSELECT | Select a square region with mouse. |
| POLYSELECT | Select a polygon shaped region with the mouse. |
| ERASE | Erase pixels in an image or in the client area. |
| OPENIMAGE | Open an image file. |
| HISTOGRAM | Create a histogram of an image or image set. |

| | |
|-------------------------------|---|
| MIRROR | Perform a horizontal mirror operation on an image. |
| FLIP | Flip an image vertically. |
| ROTATE | Rotate an image an arbitrary number of degrees. |
| SHOWRGB | Show the RGB values of a pixel, or some other operation using RGB values. |
| PICKRGB | Use the mouse to select a pixel to report RGB colors. |
| CLASSIFY | Perform a classification of an image according to pixel color. |
| NORMALIZE | Process an image to normalize the RGB histograms. |
| EQUALIZE | Process an image to equalize the RGB histograms. |
| NEGATIVE | Invert the colors of an image, creating a color negative effect. |
| GAMMACRCT | Process an image to correct for gamma. |
| IMAGEINFO | Display image information. This might be in an ImageBox or MessageBox . |
| FINGER | Use the mouse to point to or select some object. |
| HAND | Grab an object and drag it with the mouse. |
| FLOOD | Perform a flood fill operation. |
| SPRAY | Perform a color spray paint operation. |
| ZOOMUP | Zoom an image up for more detail. |
| ZOOMDN | Zoom an image down to get a synoptic view. |
| BRUSH | Select a brush. You could use ChooseColor and UseBrush commands. |
| PEN | Select a pen. You could use ChooseColor and UsePen commands. |
| SCATTRGRM | Create and display a scattergram of two images. This requires third part utilities. |
| DIGITIZE | Digitize points from the client area, or from an external device or application. |
| CDROM | Perfrom some operation using a CD-ROM. |
| SAVEALL | Save all the current open files. |
| HELPER#1 | Start a Helper Application #1. |
| HELPER#2 | Start a Helper Application #2. |
| HELPER#3 | Start a Helper Application #3. |
| HELPER#4 | Start a Helper Application #4. |
| HELPINFO | Get on-line help information. |
| Extra PiXCL 5 buttons. | Also available in geoPiXCL. |
| DIALOG | Display a dialog. |
| RECYCLE | Perform a Recycle bin operation. |
| CAMERA | Perform a digital camera operation. |
| SCANNER_1 | Perform a scanner operation. |
| GPS | Do something related to a Global Positioning System data source or set. |
| ERASE_2 | Another Erase button. |
| HELPCONTEXT | Do a Help Context operation. |
| SCANNER_2 | Perform a scanner operation. |
| SELECTSOURCE | Display the TWAIN Select Source dialog. |
| SCANNER_3 | Perform a scanner operation. |

| | |
|-------------------------------|--|
| MEASURE | Measure an image. |
| TOGGLE | Toggle some operation. |
| GeoPiXCL buttons only. | <u>These bitmaps are not available in PiXCL44 or PiXCL50.</u> |
| POINTS | Digitize points. |
| LINES | Digitize lines. |
| POLYGONS | Digitize polygons. |
| LABELS | Digitize points for text labels. |
| NDVI | N ormalized D ifference V egetation I ndex |
| AVGIMGSET | Average Image Set |
| SIGNATURE | Create a spectral signature file. |
| TRGAREAS | Define or extract training areas. |
| MLHCLAS | Maximum likelihood classifier. |
| PPDCLAS | Parallelipiped classifier. |
| PCAENHANCE | Principal components. |
| MARTAYMAP | Martin-Taylor inverse colour mapping. |
| DECORREL | Decorrelation stretch. |
| GLOBE | Global or geoid operation. |
| GEOTAG | Operations on geoTAGs in geoTIF files. |
| SHARPEN | Sharpen an image. |
| BROVEY | Brovey Transform Enhancement. |

Toolbar Button States

| | |
|----------|---|
| ENABLED | Button is functional. Most commonly used initial state. |
| CHECKED | Button is grayed and appears to be checked. |
| PRESSED | Button is grayed and appears to be pressed. |
| DISABLED | Button is grayed and non functional. |
| HIDDEN | Button is hidden. |
| NULL | Only used with SEPARATOR . |

ToolWindow Button Styles

A button's style determines how the button appears and how it responds to user input. The [STD](#) style creates a toolbar button that behaves like a standard push button that pops out when you release it. A button that has the [CHECK](#) style is similar to a standard push button, except it toggles between the pressed and nonpressed states each time the user clicks it.

| | |
|-----------|--|
| STD | Draws the standard button that pops in an out when pressed. |
| CHECK | Draws the button in a checked state. |
| STD_G | Groups buttons. Groups are separated by any non-group button |
| CHECK_G | Draws a grouped button in checked state. |
| SEPARATOR | Draws a blank between buttons |

Grouping Buttons

You can create groups of toolbar buttons by using the [STD_G](#) or [CHECK_G](#) (_G means group the buttons) styles, causing a button to stay pressed until the user chooses another button in the group. The [SEPARATOR](#) style creates a small gap between buttons. A button with this style does not receive user input.

Toolbar Customization

By holding the SHIFT key, then selecting a button, you can either delete or move that button. All buttons can be moved or deleted.

To Delete a button: drag it into the client area and release the mouse.

To Move a button: drag it to another location in the toolbar and release the mouse. The button will be inserted to the left of the button at the release location. If you have defined two sequential [SEPARATOR](#) regions in the [Toolbar](#) command, you can insert a button between them.

Double Click on the toolbar background and the standard Windows Common Controls DLL toolbar customization dialog will be displayed.

Remarks:

If you click in a toolwindow, it takes the focus, that is, the title bar changes state to indicate that it is the current window. IN addition, when you use [ToolWindow](#), the last toolwindow created gets the focus. To set the focus back to the main window, issue a [WinSetActive](#)(Title\$,Res) command, where Title\$ is the name of your application main window.

If you have moved the window around, and issue the same or different [ToolWindow\(...\)](#) command, the current toolwindows will be destroyed, and new windows created again. That is, whenever a [ToolWindow\(...\)](#) command is issued, it first clears all prior toolwindow records. This is the same method used by the [Button\(\)](#), [SetEditControl\(\)](#) and [Histogram\(\)](#) commands.

Hence, if you have, say, created two toolwindows, and you want to close one, issue a new [ToolWindow\(...\)](#) command that defines just the one toolwindow that you want. This is also the way that you can update toolwindows according to user actions.

Example:

See sample program toolbars.pxl

Related Commands:

[ChangeToolBarBtn](#) [ComboBox](#) [GetToolBarBtnStatus](#) [SetEditControl](#) [SetMenu](#) [SetPopupMenu](#), the [SetMouse](#) commands, [ToolBar](#)

Trackbar

There are many situations where you need to enter a numeric control variable, and instead of typing the value into an edit control, it is more logical to use a slider control called a Trackbar.

Trackbars can be vertical or horizontal, and come in several styles. When a Trackbar control is used,

- a) it takes the mouse and keyboard focus; and
- b) when you release the mouse, jumps to the label in your script and executes the commands found there.

You can have up to 16 Trackbars visible at any one time, and you can selectively update the range values, set the slider position, get the current slider value, and delete one or more of the controls, using the related commands, [TrackbarSetRange](#), [TrackbarSetPosition](#), [TrackbarGetValue](#) and [TrackbarRemove](#), respectively.

Syntax: [Trackbar\(x1,y1,x2,y2,mode_TOKEN, style_TOKEN, Min, Max, Frequency,Title\\$, TrackbarNumber, Label\)](#)

Parameters:

| | |
|---------------------------------|---|
| x1,y1,x2,y2 | The client area position of each trackbar. |
| mode_TOKEN | |
| F_VERT F_HORZ | Flat trackbar. |
| C_VERT C_HORZ | Client-edge (sunken) trackbar. |
| style_TOKEN | |
| TOP BOTTOM | Position of the tick marks for horizontal trackbars. |
| LEFT RIGHT | Position of the tick marks for vertical trackbars. |
| BOTH | Trackbar tick marks are on both sides. |
| Min, Max | Positive or negative range. |
| Frequency | Tick mark frequency. Default is 1. |
| Title\$ | Title of the control. If this is "", no title bar is created. If a title bar is defined, the trackbar becomes movable by click-and-dragging on the control titlebar. |
| TrackbarNumber | Zero if the operation failed, otherwise a number in the range 1-16. NOTE: The Trackbar command checks the value of TrackbarNumber . If this is non-zero, it checks if a trackbar of that ID already exists, and if it does, deletes it before creating the new trackbar. |
| Label | Jump to label for Trackbar actions. |

Remarks:

Trackbars are also controllable from the keyboard once the control has the focus.

| | |
|---|--|
| END | Moves the slider to the maximum position. |
| RIGHT or DOWN arrow key | Increments the value by 1 and adjusts the slider position. |
| LEFT or UP arrow key | Decrements the value by 1 and adjusts the slider position. |
| PAGEDOWN | Click the channel below or to the right of the slider, incrementing the slider value and position. |
| PAGEUP | Click the channel above or to the left of the slider, decrementing the slider value and position. |
| HOME | Moves the slider to the minimum position. |

For an example, see the sample program “controls.pxl”

Related Commands:

[TrackbarSetRange](#) [TrackbarSetPosition](#) [TrackbarGetValue](#) [TrackbarRemove](#)

TrackbarGetValue

Retrieves the current trackbar value

Syntax: [TrackbarGetValue](#)(*TBnumber*, *Value*)

Parameters:

[TBnumber](#) The number in range 1-16 of the Trackbar. This is the number returned by the Trackbar command.

[Value](#) The current value according to the range set in the [Trackbar](#) command.

Related Commands:

[Trackbar](#) [TrackbarSetRange](#) [TrackbarSetPosition](#) [TrackbarRemove](#)

TrackbarRemove

Closes a trackbar window, or all trackbar windows.

Syntax: [TrackbarRemove\(TBnumber\)](#)

Parameters:

TBnumber The number in range 1-16 of the Trackbar to remove. If **TBnumber** is 0, all Trackbars are removed.

Remarks:

A common programming error is to not reset **TBnumber** variables to 0 when deleting trackbars. This can result in unexpected behavior such as Trackbars being deleted or not created. For example, in the code fragment below

Make_Trackbars:

```
TrackbarRemove(0)
TBnumber6 = 0 {because it may have been used}
UseFont("Arial",7,15,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(12,4,"Red") DrawText(52,4,"Green") DrawText(108,4,"Blue")
DrawText(300,4,"Font Size")
Trackbar(0,20,49,170,C_VERT,BOTH, 0,255, 25, "Red", TBnumber3, DrawRGBr)
Trackbar(50,20,99,170,C_VERT,BOTH, 0,255, 25, "Grn", TBnumber4, DrawRGBg)
Trackbar(100,20,149,170,C_VERT,BOTH,0, 255, 25, "Blu", TBnumber5, DrawRGBb)
Trackbar(300,20,349,170,C_VERT,BOTH, 5, 36, 4, "Size",TBnumber6, DrawTestFont)
GoSub SubDrawRGB
Goto Wait_for_Input
```

Make_Single_Trackbar:

```
Trackbar(300,20,349,170,C_VERT,BOTH, 5, 36, 4, "Size",TBnumber6, DrawTestFont)
Goto Wait_for_Input
```

Delete_Single_Trackbar:

```
TrackbarRemove(TBnumber6)
Goto Wait_for_Input
```

If the code at [Make_Single_Trackbar:](#) is run, a trackbar of ID **TBnumber6** is created, which has a value 1 i.e. the first trackbar. If then the code at [Make_Trackbars:](#) is run, you have to set **TBnumber** = 0, or else the **TBnumber3** gets created and immediately removed. This is because the Trackbar command checks the value of the trackbarID variable argument. If this is non-zero, it checks if a trackbar of that ID already exists, and if it does, deletes it.

Related Commands:

[Trackbar](#) [TrackbarSetRange](#) [TrackbarSetPosition](#) [TrackbarGetValue](#)

TrackbarSetRange

Use this command to reset the range, tick mark frequency and strings of an existing Trackbar control. This is useful when you want to change only one of a group of controls.

Syntax: [TrackbarSetRange](#)(*TrackbarNumber*,*Min*,*Max*,*Frequency*)

Parameters:

[TrackbarNumber](#) The number in range 1-16 of the Trackbar.

[Min](#), [Max](#) Positive or negative range

[Frequency](#) Tick mark frequency. Default is 1.

Related Commands:

[Trackbar](#) [TrackbarSetPosition](#) [TrackbarGetValue](#) [TrackbarRemove](#)

TrackbarSetPosition

Use this command to set a new position of the slider. The range is automatically checked. If the new position is less than the minimum or greater than the maximum range value, the new position is set to the minimum or maximum position respectively.

Syntax: [TrackbarSetPosition](#)(*TrackbarNumber*, *Position*)

Parameters:

TrackbarNumber The number in range 1-16 of the Trackbar.

Position New numeric position of the slider.

Related Commands:

[Trackbar](#) , [TrackbarSetRange](#) , [TrackbarGetValue](#) , [TrackbarRemove](#)

Trim, TrimExt

These two functions trim leading and/or trailing spaces from a string.

Syntax: `Trim(String$)`

`TrimExt(String$,L|R|A)`

Parameters:

`String$` A string variable containing the text whose leading or trailing spaces you want to trim.

`L|R|A` Left, Right or All leading or trailing spaces are trimmed from `String$`.

Example:

This example solicits a line of input from the user and then trims any trailing spaces that might have inadvertently been entered.

```
Text$="Please enter your city"  
TextBox(Text$,"City",City$,Temp)  
Trim(City$)  
DrawText(10,10,City$)  
WaitInput()
```

TWAIN_AbortAllPendingXfers

It is advisable to ensure that a TWAIN device is able to transfer an image to the calling application. When programmatically controlling a device, it can happen that the TWAIN Data Source Manager (TWAIN_32.DLL) may be in an indeterminate or invalid state. Using the [TWAIN_AbortAllPendingXfers](#) command resets the Manager.

Syntax: [TWAIN_AbortAllPendingXfers](#)(*Result*)

Parameter:

Result

Usually 1. Any other value is an unexpected error condition. Often the only way to clear errors of this type is to reboot Windows. We only experienced this problem during debugging of problematic code.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_AcquireNative](#) [TWAIN_EnableSource](#)

TWAIN_AcquireNative

Acquires a single image, from the currently selected Data Source, using Native-mode transfer. It waits until the source closes (if it's modal) or forces the source closed if not. Only one image can be acquired per call.

Syntax: `TWAIN_AcquireNative(Filename,$,MODE_token,Handle)`

Parameters:

| | |
|----------------------------|---|
| Filename\$ | The name of the file that will contain the image. See Remarks below. |
| MODE_token | BW = 1-bit per pixel, B&W. GRAY = 1,4, or 8-bit grayscale. RGB = 24-bit RGB color. PALETTE = 1,4, or 8-bit palette. ANYTYPE = any of the above. |
| Handle | The bitmap handle returned. Non-zero if the operation was successful, otherwise 0. |

Remarks:

This command acquires the image and stores it in the PiXCL image list, as though it was loaded from the disk. [Filename\\$](#) is the name that the list needs to store the image, and to reference it with the [DrawBitmap](#) commands. This image is NOT written to the disk: if you need to write to disk, either subsequently use the [SaveBitmap](#) command, or initially use the [TWAIN_AcquireToFile](#) command. Once [Filename\\$](#) is stored in the PiXCL image list, it can be manipulated with any of the image processing commands, or deleted with the [FreeBitmap](#) or [FreeBitmapAll](#) commands.

Hence, if you issue a [DrawBitmap](#) command, PiXCL will display the image from memory. Only if you delete the image from the list, then issue a [DrawBitmap](#) command, PiXCL will attempt to read the image from the disk. Your program must take into account the possibility that the file may not be on the disk, or may be other image data.

[TWAIN_AcquireNative](#) encapsulates a number of the lower level commands. i.e.

- [TWAIN_LoadSourceManager](#)
- [TWAIN_OpenSourceManager](#)
- [TWAIN_OpenDefaultSource](#)
- [TWAIN_EnableSource](#)
- [TWAIN_ModalEventLoop](#) ... the image is acquired...
- [TWAIN_DisableSource](#)
- [TWAIN_CloseSource](#)
- [TWAIN_CloseSourceManager](#)
- [TWAIN_UnloadSourceManager](#)

See sample program [twaindev.pxl](#) for more information.

Related Commands:

[TWAIN_AcquireToClipboard](#) [TWAIN_AcquireToFile](#)

TWAIN_AcquireToClipboard

Acquire an image from the TWAIN device using the current device settings and passes it to the clipboard.

Syntax: [TWAIN_AcquireToClipboard\(*Result*\)](#)

Parameters:

[Result](#) 1 if the operation was successful, otherwise 0.

Remarks:

The bitmap in the clipboard can be deleted with the [ClipboardEmpty](#) command. A clipboard image can also be transferred to any other application that supports bitmap transfers. Eg. JASC PaintShopPro™.

[TWAIN_AcquireToClipboard](#) encapsulates a number of the lower level commands. I.e.

[TWAIN_LoadSourceManager](#)

[TWAIN_OpenSourceManager](#)

[TWAIN_OpenDefaultSource](#)

[TWAIN_EnableSource](#)

[TWAIN_ModalEventLoop](#) ... the image is acquired and passed to the Clipboard...

[TWAIN_DisableSource](#)

[TWAIN_CloseSource](#)

[TWAIN_CloseSourceManager](#)

[TWAIN_UnloadSourceManager](#)

Example:

See the sample application [twaindev.pxl](#).

Related Commands:

[TWAIN_AcquireToFilename](#) [TWAIN_AcquireNative](#) [ClipboardEmpty](#)

TWAIN_AcquireToFilename

Acquire an image from the TWAIN device using the current device settings, and writes it to a file.

Syntax: [TWAIN_AcquireToFilename\(*Filename*,\\$*Result*\)](#)

Parameters:

[Filename](#)\$ The name of the file that will contain the image.
[Result](#) 1 if the operation was successful, otherwise 0.

Remarks:

IF [FileName](#)\$ is a null string (""), then the SaveFile common dialog is displayed. If the file exists on disk, you will be prompted to overwrite or not.

[TWAIN_AcquireToFilename](#) encapsulates a number of the lower level commands. I.e.

- [TWAIN_LoadSourceManager](#)
- [TWAIN_OpenSourceManager](#)
- [TWAIN_OpenDefaultSource](#)
- [TWAIN_EnableSource](#)
- [TWAIN_ModalEventLoop](#) ... the image is acquired and written to the file...
- [TWAIN_DisableSource](#)
- [TWAIN_CloseSource](#)
- [TWAIN_CloseSourceManager](#)
- [TWAIN_UnloadSourceManager](#)

Example:

See the sample application [twaindev.pxl](#).

Related Commands:

[TWAIN_AcquireNative](#)[TWAIN_AcquireToClipboard](#)

TWAIN_CloseSource

Closes the open Data Source, if any. If the source is enabled, disables it first. If there is not an open source, it does nothing and returns 1.

Syntax: [TWAIN_CloseSource\(*Result*\)](#)

Parameters:

Result 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_OpenDefaultSource](#) [TWAIN_SelectSource](#)

TWAIN_CloseSourceManager

Closes the Data Source Manager, if it is open. If a source is open, disables and closes it as needed. If the Source Manager is not open does nothing and returns **@TRUE**.

Syntax: [TWAIN_CloseSourceManager\(*Result*\)](#)

Parameters:

Result 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_OpenSourceManager](#) [TWAIN_LoadSourceManager](#) [TWAIN_UnloadSourceManager](#)

TWAIN_CurrentSourceID

PIXCL 5 command. When working with multiple data sources, it can be helpful to be able to identify the source by name. This command reports the identity of the currently selected source.

Syntax: `TWAIN_CurrentSourceID(Manufacturer$, Family$, Name$, Version$)`

Parameters:

| | |
|-----------------------------|--|
| <code>Manufacturer\$</code> | The source manufacturer e.g. "Microtek" |
| <code>Family\$</code> | The product family e.g. "Scan Wizard Pro" |
| <code>Name\$</code> | The product name e.g. "Microtek Scan Wizard Pro" |
| <code>Version\$</code> | The source version e.g. "v3.20" |

Example:

This code opens the source manager, gets a source selection, then displays it in a messagebox. Once a source is selected, it remains available in the source manager until another source is selected or the application exits.

```
TWAIN_LoadSourceManager (Res)
TWAIN_SelectSource (Res)
TWAIN_OpenSourceManager (Res)
TWAIN_OpenDefaultSource (Res)
TWAIN_CurrentSourceID (Mfgr$, Family$, Name$, Version$)
Chr (13, cr$)
Res$ = Mfgr$ + cr$ + Family$ + cr$ + Name$ + cr$ + Version$
DebugMsgBox (Res$)

TWAIN_CloseSource (Res)
TWAIN_CloseSourceManager (Res)
TWAIN_UnLoadSourceManager (Res)
```

Related Commands:

[TWAIN_OpenSourceManager](#) [TWAIN_EnumSource](#) [TWAIN_GetDefaultID](#)

TWAIN_DisableSource

Disables the open Data Source, if any. This closes the source's user interface. If there is not an enabled source, does nothing and returns 1.

Syntax: [TWAIN_DisableSource\(*Result*\)](#)

Parameters:

[Result](#) 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_EnableSource](#)

TWAIN_EnableSource

Instructs the current TWAIN source to get ready to acquire an image. All TWAIN devices have a Data Source (often called a driver), and is a file with a **.DS** extension stored in the Windows directory (typically **c:\windows**). Some Data Sources may be stored in subdirectories as well. The Data Source is the interface between the device itself, which may be connected to a COM port, parallel port, SCSI bus or Universal Serial Bus (USB), and the Windows TWAIN_32.DLL, which is the Data Source Manager. You can set the current device with the [TWAIN_SelectSource](#) command, and this selection is stored by Windows. That is, you can select a source in a PiXCL application, shut it down, then restart it, and the previous TWAIN device selection is still valid. If you shut down Windows and restart, the default TWAIN device is the first device in the list displayed by the [TWAIN_SelectSource](#) command.

Syntax: [TWAIN_EnableSource](#)(*Result*)

Parameters:

Result 1 if the operation was successful and an image was acquired, otherwise 0.

Remarks:

A series of commands is necessary prior to using this one. For example, [TWAIN_AcquireNative](#) encapsulates a number of the lower level commands. i.e.

- [TWAIN_LoadSourceManager](#)
- [TWAIN_OpenSourceManager](#)
- [TWAIN_OpenDefaultSource](#)
- [TWAIN_EnableSource](#)
- [TWAIN_ModalEventLoop](#) ... the image is acquired...
- [TWAIN_DisableSource](#)
- [TWAIN_CloseSource](#)
- [TWAIN_CloseSourceManager](#)
- [TWAIN_UnloadSourceManager](#)

A PiXCL application will use a sequence like the above.

Example:

[TWAIN_EnableSource](#) is generally followed by [TWAIN_ModalEventLoop](#). See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_ModalEventLoop](#) [TWAIN_AcquireNative](#) [TWAIN_DisableSource](#) [TWAIN_SelectSource](#)

TWAIN_EnableUI

A TWAIN data source has a 'hide source user interface' flag, which is cleared initially, but if you set it, when a source is enabled it will be asked to hide its user interface. Note that this is only a request - **some sources will ignore it!** This affects [TWAIN_AcquireNative](#), [TWAIN_AcquireToClipboard](#), and [TWAIN_EnableSource](#).

If the user interface is hidden, you will probably want to set at least some of the basic acquisition parameters yourself, using [TWAIN_SetCurrentUnits](#), [TWAIN_SetBitDepth](#), [TWAIN_SetPixelFormat](#) and [TWAIN_SetCurrentRes](#) below. Some data sources also require that brightness and contrast be initialized as well with a [TWAIN_SetCapability](#) command before an image is acquired. An example is the Microtek ScanWizardPro © product that does a calibration pass before attempting image acquisition.

Syntax: [TWAIN_EnableUI](#)(Mode)

Parameter:

[Mode](#) 1 or **@TRUE** (the default) to enable the user interface, or 0 or **@FALSE** to disable it.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_SetCurrentUnits](#) [TWAIN_SetBitDepth](#) [TWAIN_SetPixelFormat](#) [TWAIN_SetCurrentRes](#)

TWAIN_EnumSource

PIXCL 5 command. Instead of displaying the Select Source dialog, you can enumerate the installed sources and later use the [TWAIN_OpenSpecificSource](#) command.

Syntax: [TWAIN_EnumSource\(DS_List\\$,Number\)](#)

Parameters:

[DS_List\\$](#) A carriage-return delimited list of the TWAIN Data Source product names installed and recognized by the system. This is the same list of names that appears in the Select Source dialog.

[Number](#) The number of Data Sources located in the TWAIN and TWAIN_32 subdirectories.

Related Commands:

[TWAIN_SelectSource](#) [TWAIN_OpenSpecificSource](#)

TWAIN_GetBitDepth

Get the current bit depth, which can depend on the current PixelType. Bit depth is bits per color channel e.g. 24-bit RGB has bit depth 8. If anything goes wrong, this function returns 0.

Syntax: [TWAIN_GetBitDepth\(*Result*\)](#)

Parameters:

[Result](#) Bits per color channel if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_SetBitDepth](#)

TWAIN_GetBitmapParams

Get the parameters of the bitmap acquired by the TWAIN source device..

Syntax: [TWAIN_GetBitmapParams\(Handle,Lines,Pixels,Bits,Colors\)](#)

Parameters:

| | |
|------------------------|---|
| Handle | The handle of the current bitmap, returned from TWAIN_AcquireNative . |
| Lines | Number of lines in the bitmap. |
| Pixels | Number of pixels per line. |
| Bits | Number of bits per pixel. |
| Colors | Number of colors in the colormap. |

Example:

See the sample application [twaindev.pxl](#).

Related Commands:

[GetBitmapDim](#) [TWAIN_AcquireNative](#)

TWAIN_GetCapability

PIXCL 5 command Any supported TWAIN device capability can be acquired with this command. Note that all devices do not support all capabilities, and the allowable range of values don't necessarily apply to all devices. In addition, TWAIN data sources (i.e. the driver) are often badly behaved, and don't support all capabilities correctly.

There are over 150 capabilities defined in the TWAIN specification, and we have implemented most of the commonly used ones in PIXCL. If you have a TWAIN device and a desired capability token is not listed here, please contact VYSOR Technical Support at <http://www.vysor.com>, and we'll put into the next maintenance release. This typically takes 3-5 business days.

Syntax: TWAIN_GetCapability(Cap_TOKEN, Value|Value&|Value\$)

Parameters:

Cap_TOKEN

One of the following tokens:

AUTOBRIGHT

Use **Value**. Returns 0 or 1 if auto brightness correction is supported.

BRIGHTNESS

Use **Value** or **Value&**. Default value 0, settable in range -1000 to + 1000.

COMPRESSION

Use **Value**. Returns 0 if compression is not supported, otherwise

1 = PACKBITS Usually a TIFF format mode.

2 = GROUP31D Follows CCITT spec

3 = GROUP31DEOL Follows CCITT spec

4 = GROUP32D Follows CCITT spec

5 = GROUP4 Follows CCITT spec

6 = JPEG Use capability for more info.

7 = LZW Must license from Unisys and IBM to use.

8 = JBIG For Bitonal images.

9 = PNG Portable Network Graphics

10 = RLE4 4 bits / pixel

11 = RLE8 8 bits / pixel

12 = BITFIELDS

CONTRAST

Use **Value** or **Value&**. Default value 0, theoretically settable in range -1000 to + 1000.

EXPOSURETIME

Use **Value**. Returns the exposure time in seconds. Must be < 0.

FILTER

Use **Value**. Returns the colour characteristics of the subtractive filter applied to the image. Values are

0 = Red 1 = Green 2 = Blue 3 = None 4 = White

5 = Cyan 6 = Magenta 7 = Yellow 8 = Black

FLASHUSED

Use **Value**. Returns 1 if a flash was used to acquire the image, otherwise 0.

GAMMA

Use **Value&**. Default is 2.20, settable to any float value.

HIGHLIGHT

Use **Value**. Specifies the lightest highlight value in the image, in the range 0-255. All higher values are clipped at this level.

IMAGEFILEFORMAT

Use **Value**.

LAMPSTATE

Use **Value**. A return of 1 indicates that the lamp is on or should be set to on.

LIGHTPATH

Use **Value**. Returns 0 for reflective mode, and 1 for transmissive.

LIGHTSOURCE

Use **Value**. Describes the current characteristics of the light source in use, Values are

0 = Red 1 = Green 2 = Blue 3 = None 4 = White 5 = UV 6 = IR

ORIENTATION

Use **Value**. Returns the page orientation.

PHYSICALWIDTH

Use **Value**. The maximum width of an image that can be acquired, in the current units value.

PHYSICALHEIGHT

Use **Value**. The maximum height of an image that can be acquired, in the current units value.

SHADOW

Use **Value**. Specifies the darkest shadow value in the image, in the range 0-255. All lower

| | |
|---------------------------|---|
| | values are clipped at this level. |
| XNATIVERESOLUTION | Use <i>Value</i> . Maximum X resolution of the device optics. |
| YNATIVERESOLUTION | Use <i>Value</i> . Maximum Y resolution of the device optics. |
| XRESOLUTION | Use <i>Value</i> . Current X scan resolution. |
| YRESOLUTION | Use <i>Value</i> . Current Y scan resolution. |
| MAXFRAMES | Use <i>Value</i> . |
| TILES | Use <i>Value</i> . |
| BITORDER | Use <i>Value</i> . Specifies which bit per pixel is the most significant. Returns 0 for LSB first and 1 for MSB first (the default). |
| BITORDERCODES | Use <i>Value</i> . |
| CCITTKFACTOR | Use <i>Value</i> . |
| PIXELFLAVOR | Use <i>Value</i> . Returns 0 for pixel value 0 = black, 1 for pixel value 0 = white. |
| PIXELFLAVORCODES | Use <i>Value</i> . |
| PLANARCHUNKY | Use <i>Value</i> |
| ROTATION | Use <i>Value</i> |
| SUPPORTEDSIZES | Use <i>Value</i> |
| THRESHOLD | Use <i>Value</i> |
| XSCALING | Use <i>Value</i> &. Default is 1.0, can be any value > 0.0. |
| YSCALING | Use <i>Value</i> &. Default is 1.0, can be any value > 0.0. |
| JPEGPIXELTYPE | Use <i>Value</i> |
| TIMEFILL | Use <i>Value</i> |
| BITDEPTH | Use <i>Value</i> . Returns the bits per pixel per channel. Must be >= 1. Most commonly used value is 8, but 4 and 16 are often supported too. |
| BITDEPTHREDUCTION | Use <i>Value</i> |
| UNDEFINEDIMAGESIZE | Use <i>Value</i> |
| IMAGEDATASET | Use <i>Value</i> |
| EXTIMAGEINFO | Use <i>Value</i> |
| MINIMUMHEIGHT | Use <i>Value</i> |
| MINIMUMWIDTH | Use <i>Value</i> |
| FLIPROTATION | Use <i>Value</i> . |
| SERIALNUMBER | Use <i>Value</i> \$. Returns the serial number of the device, or an empty string if not supported. |
| AUTHOR | Use <i>Value</i> \$. Returns the image author, or an empty string if not supported. Sometimes is the name of the developer. |

Value|&|\$ The value to get, as the appropriate variable type. Unsupported capabilities return values of 0, 0.0 or an empty string. If you get an unexpected 0 from what you might expect to be an integer value, try a floating point value instead.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_SetCapability](#)

TWAIN_GetCompression

PIXCL 5 command. Many TWAIN devices, particularly scanners, support data compression as an option for the acquired image data. You can use this command to query the current mode.

Syntax: `TWAIN_GetCompression(CompressMode)`

Parameter:

CompressMode

This returned value will be one of the following.

- 0 = **NONE** A device that does not support compression always reports this.
- 1 = **PACKBITS** Usually a TIFF format mode.
- 2 = **GROUP31D** Follows CCITT spec
- 3 = **GROUP31DEOL** Follows CCITT spec
- 4 = **GROUP32D** Follows CCITT spec
- 5 = **GROUP4** Follows CCITT spec
- 6 = **JPEG** Use capability for more info.
- 7 = **LZW** Must license from Unisys and IBM to use.
- 8 = **JBIG** For Bitonal images.
- 9 = **PNG** Portable Network Graphics
- 10 = **RLE4** 4 bits / pixel
- 11 = **RLE8** 8 bits / pixel
- 12 = **BITFIELDS**

Example:

See the sample application `twaindev.pxl`.

Related Commands:

[TWAIN_GetCapability](#) [TWAIN_SetCompression](#)

TWAIN_GetCurrentRes

Ask the source for the current device image acquisition resolution. Resolution is in dots per current unit. See also [TWAIN_GetCurrentUnits](#).

Syntax: [TWAIN_GetCurrentRes\(*Result*\)](#)

Parameters:

[Result](#) Non-zero if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_GetCurrentUnits](#)

TWAIN_GetCurrentUnits

Ask the source for its current unit of measure. If anything goes wrong, this function just returns 0, which is the default, signifying inches.

Syntax: [TWAIN_GetCurrentUnits\(*Result*\)](#)

Parameters:

Result Possible values are:
0 = **INCHES**
1 = **CENTIMETERS**
2 = **PICAS**
3 = **POINTS**
4 = **TWIPS**
5 = **PIXELS**

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_GetCurrentRes](#)

TWAIN_GetFrame

PIXCL 5 command. Many devices, particularly scanners, have the option within the user interface to set a framed area from which the image is to be acquired. The current frame settings can be queried under program control while in State 4, assuming the Data Source allows it. The frame coordinates are expressed in the current **UNITS** as set by [TWAIN_SetCurrentUnits](#) or retrieved with [TWAIN_GetCurrentUnits](#).

Syntax: [TWAIN_GetFrame\(x1&,y1&,x2&, y2&,Result\)](#)

Parameters:

[x1&,y1&,x2&, y2&](#) The current frame coordinates.

[Result](#) 1 if the operation was successful, otherwise 0.

Remarks:

The frame coordinates are directly related to the current scanning resolution, not the native resolution of the scanner. It is your responsibility to take into account both factors in your application. For example, consider a scanner that has a 10 inch square scanning surface. At 300 pixels per inch, the X and Y axis coordinates range from 0 to 3000, while at 50 pixels per inch, the range is 0 to 500. Let's say that you set frame coordinates of (100,100) and (600,700). At 300 pixels per inch a correct image will be scanned, while at 50 pixels per inch, you may get an image (or a badly behaved Data Source may crash itself and/or PiXCL) but it will be scrambled, often due to scan line wrap around.

Related Commands:

[TWAIN_SetFrame](#)

TWAIN_GetDefaultID

PIXCL 5 command. When working with multiple data sources, it can be helpful to be able to identify the source by name. This command reports the identity of the currently selected source without opening the default source.

Syntax: `TWAIN_GetDefaultID(Manufacturer$, Family$, Name$, Version$)`

Parameters:

| | |
|-----------------------------|--|
| <code>Manufacturer\$</code> | The source manufacturer e.g. "Microtek" |
| <code>Family\$</code> | The product family e.g. "Scan Wizard Pro" |
| <code>Name\$</code> | The product name e.g. "Microtek Scan Wizard Pro" |
| <code>Version\$</code> | The source version e.g. "v3.20" |

Example:

This code opens the source manager, gets a source selection, then displays it in a messagebox. Once a source is selected, it remains available in the source manager until another source is selected or the application exits.

```
TWAIN_LoadSourceManager (Res)
TWAIN_SelectSource (Res)
TWAIN_OpenSourceManager (Res)
TWAIN_GetDefaultID (Mfgr$, Family$, Name$, Version$)
Chr (13, cr$)
Res$ = Mfgr$ + cr$ + Family$ + cr$ + Name$ + cr$ + Version$
DebugMsgBox (Res$)

TWAIN_CloseSourceManager (Res)
TWAIN_UnLoadSourceManager (Res)
```

Related Commands:

[TWAIN_OpenSourceManager](#) [TWAIN_EnumSource](#) [TWAIN_CurrentSourceID](#)

TWAIN_GetPixelFormat

Ask the source for the current pixel type. If anything goes wrong (it shouldn't), this function returns 0.

Syntax: [TWAIN_GetPixelFormat\(*Result*\)](#)

Parameters:

Result Possible values are

- 0 = **BW**
- 1 = **GRAY**
- 2 = **RGB**
- 3 = **PALETTE**
- 4 = **CMY**
- 5 = **CMYK**
- 6 = **YUV**
- 7 = **YUVK**
- 8 = **CIEXYZ**

Example:

See the sample application [twaindev.pxl](#).

Related Commands:

[TWAIN_SetPixelFormat](#)

TWAIN_GetState

TWAIN device communication protocol with the host application and **TWAIN_32.DLL** uses what are called States. Different states are generated as the acquisition operation proceeds. The [TWAIN_GetState](#) command is used when you need to handle low level commands, rather than high level commands like the [TWAIN_Acquire*](#) commands. Low level commands like [TWAIN_SetBitDepth](#) only work at State 4 or higher.

Syntax: [TWAIN_GetState\(CurrentState\)](#)

Parameters:

[CurrentState](#) If the operation was not successful returns 0, otherwise

- 1 = source manager not loaded
- 2 = source manager loaded
- 3 = source manager open
- 4 = source open but not enabled
- 5 = source enabled to acquire
- 6 = image ready to transfer
- 7 = image in transit

Related Commands:

[TWAIN_IsAvailable](#)

TWAIN_IsAvailable

Call this function at any time to find out if TWAIN is installed on the system. It takes a little time on the first call, after that it's fast, just testing a flag. Most standard Windows installations already include TWAIN_32.DLL. Also, when you install a TWAIN device such as a scanner, it will generally check if TWAIN is installed, and if not, install it.

Syntax: [TWAIN_IsAvailable\(*Result*\)](#)

Parameters:

[Result](#) Returns 1 if the TWAIN Source Manager (TWAIN_32.DLL) is installed and can be loaded, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_LoadSourceManager](#)

TWAIN_LoadSourceManager

Finds and loads the Data Source Manager, TWAIN_32.DLL. If the Source Manager is already loaded, the command does nothing and returns 1. This can fail if TWAIN_32.DLL is not installed in the Windows directory, or if the library cannot load for some reason (e.g. insufficient memory) or if TWAIN_32.DLL has been corrupted.

If you have multiple TWAIN devices e.g. two scanners, and your application has to be able to select between devices, it a good idea to load and open the Source Manager once at start up, and shut it down on exit.

Syntax: [TWAIN_LoadSourceManager\(Result\)](#)

Parameters:

[Result](#) 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_OpenSourceManager](#) [TWAIN_UnloadSourceManager](#) [TWAIN_IsAvailable](#)

TWAIN_ModalEventLoop

When you are programmatically controlling a TWAIN device, you have to have a way to actually acquire the image. If you are using [TWAIN_Acquire*](#) commands, this is encapsulated within the command. When manually controlling the state of the TWAIN device, you must use this command.

Syntax: [TWAIN_ModalEventLoop\(ImageName\\$,Handle\)](#)

Parameters:

[Filename\\$](#) The name of the file or image list entry that will contain the image. The image format is the current setting.

[Handle](#) The bitmap handle returned. Non-zero if the operation was successful, otherwise 0.

Remarks:

Almost all TWAIN data sources display a message dialog “**Transferring image to <name>**”. You can hit the Escape key to abort the scan in progress. [Handle](#) will return 0, and can be used to check for the abort operation. To set **<name>** use [TWAIN_RegisterApp](#).

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_UnloadSourceManager](#) [TWAIN_IsAvailable](#)
[TWAIN_EnableSource](#) [TWAIN_RegisterApp](#)

TWAIN_OpenDefaultSource

This opens the source selected in the Select Source dialog. If a source is already open, the command does nothing and returns 1. Fails if the source manager is not loaded and open.

Syntax: [TWAIN_OpenDefaultSource\(*Result*\)](#)

Parameters:

[Result](#) 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_LoadSourceManager](#) [TWAIN_UnloadSourceManager](#) [TWAIN_IsAvailable](#)

TWAIN_OpenSourceManager

Opens the Data Source Manager, if not already open. If the Source Manager is already open, does nothing and returns 1. This call will fail if the Source Manager is not loaded.

Syntax: [TWAIN_OpenSourceManager\(Result\)](#)

Parameters:

Result 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_LoadSourceManager](#) [TWAIN_UnloadSourceManager](#) [TWAIN_IsAvailable](#)

TWAIN_OpenSpecificSource

PIXCL 5 command. If you know the name of the Data Source you want to open, you can use this command and bypass the Select Source dialog. Please note that this does **NOT** set the Default Data Source that is displayed by the [TWAIN_SelectSource](#) command.

Syntax: [TWAIN_OpenSpecificSource\(DS_Name\\$,Result\)](#)

Parameters:

| | |
|---------------------------|--|
| DS_Name\$ | The specific Data Source name. |
| Result | 1 if the source was opened, otherwise 0. |

Related Commands:

[TWAIN_EnumSource](#) [TWAIN_OpenDefaultSource](#) [TWAIN_SelectSource](#)

TWAIN_PxlVersion

Get the version number of the **PXLtwain.dll** library file.

Syntax: `TWAIN_PxlVersion(Result)`

Parameters:

Result The version number of the PXLtwain.dll. This will be ≥ 500 .

Related Commands:

None.

TWAIN_RegisterApp

TWAIN_RegisterApp can be called *AS THE FIRST CALL*, to register the application. If this function is not called, the application is given a 'generic' registration by PXLtwain.

Registration only provides this information to the TWAIN Source Manager and any sources you may open - it is used by some sources to give special treatment to certain applications.

Syntax: TWAIN_RegisterApp(*Major,Minor,Lang,Country,Version\$,groupTOKEN,Mfg\$,Fam\$,Prod\$*)

Parameters:

| | |
|---------------------|---|
| <i>Major,Minor</i> | Version numbers. E.g. 4, 40 |
| <i>Lang,Country</i> | Number Codes. Consult your Windows documentation for the correct codes for your language and country, or just set them to 0. |
| <i>Version\$</i> | String e.g. "4.40" |
| <i>groupTOKEN</i> | One of the following to mask the devices selectable from TWAIN_SelectSource. IMAGE Most commonly used, for image acquisition devices only. CONTROL Select control devices only. AUDIO Occasionally used in recent devices with audio capture. ALL All of the above. |
| <i>Mfg\$</i> | e.g. "VYSOR" |
| <i>Fam\$</i> | e.g. "PIXCL Tools" |
| <i>Prod\$</i> | e.g. "Image Scanner Application". For example, Microtek scanners with Scan Wizard™ or Scan Wizard Pro™ display a dialog box when the scan is occurring, with the message "Transferring image to <name>". This <name> can be replaced with Prod\$. |

Example:

See the sample application **twaindev.pxl**.

Related Commands:

None.

TWAIN_SelectSource

This is the routine to call when the user chooses the standard **"Select Source..."** menu command from your application's File menu. The TWAIN specification calls for this feature to be available in your user interface, preferably as described.

Syntax (PIXCL 4): [TWAIN_SelectSource\(Result\)](#)

Syntax (PIXCL 5): [TWAIN_SelectSource\(HighLight | HighLight\\$, Result\)](#)

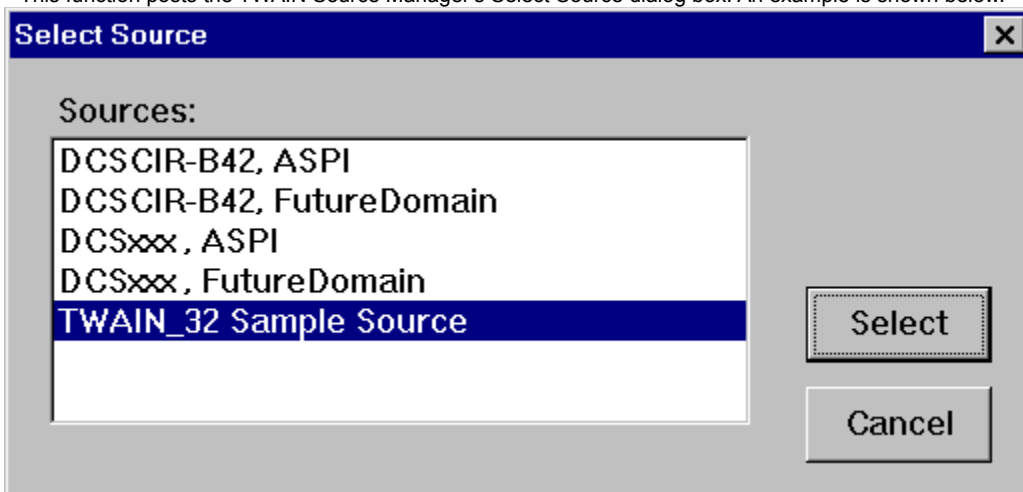
Parameters:

[HighLight | HighLight\\$](#) The 0-based Data Source index that is highlighted, or the name string of the Data Source. In the example dialog shown below, Highlight = 4, Highlight\$ = "TWAIN_32 Sample Source".

[Result](#) 1 if the operation was successful, otherwise 0.

Remarks:

This function posts the TWAIN Source Manager's Select Source dialog box. An example is shown below.



A return of 1 indicates OK, 0 indicates one of the following:

- The user cancelled the dialog
- The Source Manager found no data sources installed
- There was a failure before the Select Source dialog could be posted

Only sources that can return images are displayed, and the current default source will be highlighted initially, unless the index or source name is specified. In the standard implementation of **"Select Source..."**, your application doesn't need to do anything except make this one call.

If you want to be meticulous, disable your **"Acquire"** and **"Select Source..."** menu items or buttons if [TWAIN_IsAvailable](#) returns 0.

Note#1: If only one TWAIN device is installed on a system, it is selected automatically, so there is no need for the user to do Select Source. You should not require your users to do Select Source before Acquire.

Note#2: The selection you make on Select is retained within Windows, and becomes the Default Data Source for ALL applications.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_IsAvailable](#)

TWAIN_SetBitDepth

This command tries to set the per-channel bit depth for the current acquisition pixel type. For most applications, a bit depth of 8 is the default. Some data sources accept 1 and 16. Hence, if your pixel type is **GRAY** and bit depth is 8, then the bits per pixel of your image will be 8. If your pixel type is **RGB** and bit depth is 8, then the bits per pixel of your image will be 24. If your pixel type is **RGB** and bit depth is 16, then the bits per pixel of your image will be 48. If your pixel type is **CMYK** and bit depth is 8, then the bits per pixel of your image will be 32.

Syntax: `TWAIN_SetBitDepth(Depth|Result)`

Parameters:

Depth

Must be a variable. The desired bit depth for scanning. The same

Result

variable returns 1 if the operation was successful, otherwise 0. The bit depth must be appropriate to the pixel type last set with `TWAIN_SetPixelFormat`. If the bit depth is not supported under the current pixel type, *Result* returns 0, and the bitdepth is unchanged.

Example:

See the sample application `twaindev.pxl`.

Related Commands:

[TWAIN_GetBitDepth](#) `TWAIN_SetPixelFormat`

TWAIN_SetCapability

PIXCL 5 command. Any supported TWAIN device capability can be set with this command.

There are over 150 capabilities defined in the TWAIN specification, and we have implemented most of the commonly used ones in PiXCL. If you have a TWAIN device and a desired capability token is not listed here, please contact VYSOR Technical Support at <http://www.vysor.com>, and we'll put into the next maintenance release. This typically takes 3-5 business days.

Syntax: TWAIN_SetCapability(Cap_TOKEN, Value|Value&|Value\$)

Parameters:

| | |
|--------------------------|---|
| Cap_TOKEN | One of the following tokens: |
| AUTOBRIGHT | Use Value. Returns 0 or 1 if auto brightness correction is supported. |
| BRIGHTNESS | Use Value or better Value&. Default value 0, settable in range -1000 to + 1000. |
| COMPRESSION | Use Value. Returns 0 if compression is not supported, otherwise 1 = PACKBITS Usually a TIFF format mode. 2 = GROUP31D Follows CCITT spec 3 = GROUP31DEOL Follows CCITT spec 4 = GROUP32D Follows CCITT spec 5 = GROUP4 Follows CCITT spec 6 = JPEG Use capability for more info. 7 = LZW Must license from Unisys and IBM to use. 8 = JBIG For Bitonal images. 9 = PNG Portable Network Graphics 10 = RLE4 4 bits / pixel 11 = RLE8 8 bits / pixel 12 = BITFIELDS |
| CONTRAST | Use Value or better Value&. Default value 0, theoretically settable in range -1000 to + 1000. |
| EXPOSURETIME | Use Value. Returns the exposure time in seconds. Must be < 0. |
| FILTER | Use Value. Returns the colour characteristics of the subtractive filter applied to the image. Values are 0 = Red 1 = Green 2 = Blue 3 = None 4 = White 5 = Cyan 6 = Magenta 7 = Yellow 8 = Black |
| FLASHUSED | Use Value. Returns 1 if a flash was used to acquire the image, otherwise 0. |
| GAMMA | Use Value&. Default is 2.20, settable to any float value. |
| HIGHLIGHT | Use Value. Specifies the lightest highlight value in the image, in the range 0-255. All higher values are clipped at this level. |
| IMAGEFILEFORMAT | Use Value. |
| LAMPSTATE | Use Value. A return of 1 indicates that the lamp is on or should be set to on. |
| LIGHTPATH | Use Value. Returns 0 for reflective mode, and 1 for transmissive. |
| LIGHTSOURCE | Use Value. Describes the current characteristics of the light source in use, Values are 0 = Red 1 = Green 2 = Blue 3 = None 4 = White 5 = UV 6 = IR |
| ORIENTATION | Use Value. Returns the page orientation. |
| PHYSICALWIDTH | Use Value. The maximum width of an image that can be acquired, in the current units value. |
| PHYSICALHEIGHT | Use Value. The maximum height of an image that can be acquired, in the current units value. |
| SHADOW | Use Value. Specifies the darkest shadow value in the image, in the range 0-255. All lower values are clipped at this level. |
| XNATIVERESOLUTION | Use Value. Maximum X resolution of the device optics. |

| | |
|---------------------------|---|
| YNATIVERESOLUTION | Use <i>Value</i> . Maximum Y resolution of the device optics. |
| XRESOLUTION | Use <i>Value</i> . Current X scan resolution. |
| YRESOLUTION | Use <i>Value</i> . Current Y scan resolution. |
| MAXFRAMES | Use <i>Value</i> . |
| TILES | Use <i>Value</i> . |
| BITORDER | Use <i>Value</i> . Specifies which bit per pixel is the most significant. Returns 0 for LSB first and 1 for MSB first (the default). |
| BITORDERCODES | Use <i>Value</i> . |
| CCITTKFACTOR | Use <i>Value</i> . |
| PIXELFLAVOR | Use <i>Value</i> . Returns 0 for pixel value 0 = black, 1 for pixel value 0 = white. |
| PIXELFLAVORCODES | Use <i>Value</i> . |
| PLANARCHUNKY | Use <i>Value</i> . |
| ROTATION | Use <i>Value</i> . |
| SUPPORTEDSIZES | Use <i>Value</i> . |
| THRESHOLD | Use <i>Value</i> . |
| XSCALING | Use <i>Value</i> &. Default is 1.0, can be any value > 0.0. |
| YSCALING | Use <i>Value</i> &. Default is 1.0, can be any value > 0.0. |
| JPEGPIXELTYPE | Use <i>Value</i> . |
| TIMEFILL | Use <i>Value</i> . |
| BITDEPTH | Use <i>Value</i> . Returns the bits per pixel per channel. Must be >= 1. Most commonly used value is 8, but 4 and 16 are often supported too. |
| BITDEPTHREDUCTION | Use <i>Value</i> . |
| UNDEFINEDIMAGESIZE | Use <i>Value</i> . |
| IMAGEDATASET | Use <i>Value</i> . |
| EXTIMAGEINFO | Use <i>Value</i> . |
| MINIMUMHEIGHT | Use <i>Value</i> . |
| MINIMUMWIDTH | Use <i>Value</i> . |
| FLIPROTATION | Use <i>Value</i> . |
| <i>Value</i> & &\$ | The value to set, as the appropriate variable type. |

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_GetCapability](#)

TWAIN_SetCompression

PIXCL 5 command. Many TWAIN devices, particularly scanners, support data compression as an option for the acquired image data. You can use this command to set the current mode.

Syntax: `TWAIN_SetCompression(CompressMode)`

Parameter:

CompressMode

This value must be one of the following.

- 0 = **NONE** A device that does not support compression always reports this.
- 1 = **PACKBITS** Usually a TIFF format mode.
- 2 = **GROUP31D** Follows CCITT spec
- 3 = **GROUP31DEOL** Follows CCITT spec
- 4 = **GROUP32D** Follows CCITT spec
- 5 = **GROUP4** Follows CCITT spec
- 6 = **JPEG** Use capability for more info.
- 7 = **LZW** Must license from Unisys and IBM to use.
- 8 = **JBIG** For Bitonal images.
- 9 = **PNG** Portable Network Graphics
- 10 = **RLE4** 4 bits / pixel
- 11 = **RLE8** 8 bits / pixel
- 12 = **BITFIELDS**

Example:

See the sample application `twaindev.pxl`.

Related Commands:

[TWAIN_GetCompression](#)

TWAIN_SetCurrentRes

Try to set the current resolution for acquisition. Resolution is expressed in dots per current unit. See [TWAIN_GetCurrentUnits](#). This is only allowed in State 4 (TWAIN_SOURCE_OPEN) as returned by [TWAIN_GetState](#). Note: The source may select this resolution, but don't assume it will.

Syntax: [TWAIN_SetCurrentRes\(Resolution|Result\)](#)

Parameters:

[Resolution|Result](#) Must be a variable with a non-zero value. Consult your data source documentation for the set of correct values. Typical values are 300, 600, 1200, 2400. Returns 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_GetCurrentUnits](#) [TWAIN_GetState](#)

TWAIN_SetCurrentUnits

Set the TWAIN device current units type. In most cases you will use [PIXELS](#).

Syntax: [TWAIN_SetCurrentUnits\(UNIT_token,Result\)](#)

Parameters:

[UNIT_token](#) INCHES, CENTIMETERS, PICAS, POINTS, TWIPS, PIXELS
[Result](#) 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_GetCurrentUnits](#)

TWAIN_SetFrame

PIXCL 5 command. Many devices, particularly scanners, have the option within the user interface to set a framed area from which the image is to be acquired. This frame can also be set under program control while in TWAIN State 4 only, assuming the Data Source allows it. The frame coordinates are expressed in the current UNITS as set by [TWAIN_SetCurrentUnits](#) or retrieved with [TWAIN_GetCurrentUnits](#).

Syntax: [TWAIN_SetFrame\(x1&,y1&,x2&, y2&,Result\)](#)

Parameters:

[x1&,y1&,x2&, y2&](#) The desired new frame coordinates.
[Result](#) 1 if the operation was successful, otherwise 0.

Remarks:

The frame coordinates are directly related to the current scanning resolution, not the native resolution of the scanner. It is your responsibility to take into account both factors in your application. For example, consider a scanner that has a 10 inch square scanning surface. At 300 pixels per inch, the X and Y axis coordinates range from 0 to 3000, while at 50 pixels per inch, the range is 0 to 500. Let's say that you set frame coordinates of (100,100) and (600,700). At 300 pixels per inch a correct image will be scanned, while at 50 pixels per inch, you may get an image (or a badly behaved Data Source may crash itself and/or PIXCL) but it will be scrambled, often due to scan line wrap around.

Related Commands:

[TWAIN_GetFrame](#)

TWAIN_SetPixelFormat

Try to set the current pixel type for acquisition. This is only allowed in State 4, as returned by [TWAIN_GetState](#). The source may select this pixel type, but don't assume it will.

Syntax: [TWAIN_SetPixelFormat\(TYPE_token,Result\)](#)

Parameters:

[TYPE_token](#) BW, GRAY, RGB, PALETTE, CMY, CMYK, YUV, YUVK, CIEXYZ.

Result 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_GetPixelFormat](#), [TWAIN_GetState](#)

TWAIN_UnloadSourceManager

Unloads the TWAIN device manager and sets the State to 1.

Syntax: [TWAIN_UnloadSourceManager\(*Result*\)](#)

Parameters:

[Result](#) 1 if the operation was successful, otherwise 0.

Example:

See the sample application **twaindev.pxl**.

Related Commands:

[TWAIN_LoadSourceManager](#)

UCase

Converts a string to uppercase.

Syntax: `UCase(String$)`

Parameter:

`String$` The string to convert.

Example:

This program reads the text on the Clipboard, converts it to uppercase, and then writes it back out to the Clipboard.

```
ClipboardGet (String$, Result)  
UCase (String$)  
ClipboardPut (String$, Result)
```

Related Command:

[LCase](#)

UninstallColorProfile

PIXCL 5 command: Windows 98 or later and Windows 2000 include colour management functions using the standardised profiles in the International Colour Consortium (ICC) format. Colour profiles are stored in the **c:\windows\system\color** directory, and have to be made known to Windows before they can be used. At times it may be required to uninstall an existing profile and install a new one of the same name.

Syntax: [UninstallColorProfile\(ProfileName\\$,Result\)](#)

Parameters:

[ProfileName\\$](#) The name of a colour profile, extension **.icm**.
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[InstallColorProfile](#)

UnpackRGB

PIXCL 5 command. It can often be handy to unpack RGB color values from an integer array variable into integers.

Syntax: `UnpackRGB(PackedColour, Red, Green, Blue)`

Parameters:

PackedColour The packed colour value, in hex format, 0x00bbggrr

Red, *Green*, *Blue* Integer values for the defined colour.

Related Commands:

[PackRGB](#) [PackRGBA](#)

UnpackRGBA

PIXCL 5 command. It can often be handy to unpack RGB and Alpha color values from an integer array variable into integers.

Syntax: `UnpackRGBA(PackedColour, Red, Green, Blue, Alpha)`

Parameters:

PackedColour The packed colour value, in hex format, 0x00bbgrr

Red, *Green*, *Blue*, *Alpha* Integer values for the defined colour.

Related Commands:

[PackRGBA](#) [PackRGB](#)

UnregisterUserCmds

Use this command to unregister all user commands. If you use an extension command after this command, a syntax error occurs unless the command or command library have been re-registered. If the DLL that registered the command set is loaded, it is automatically unloaded. Note that if the DLL calls other large DLLs that reference special hardware drivers, the unload process may take some seconds.

Syntax: [UnregisterUserCmds\(Count\)](#)

Parameter:

[Count](#) The number of commands unregistered.

Related Commands:

[CountRegdUserCmds](#) [RegisterUserCommand](#) [RegisterExtLibCmdSet](#)

UpdateHistogram

Histogram windows can be updated without recreating the window. This is handy when there are multiple Histograms displayed in the application. If the image has been changed with one of the image processing commands, or you want to change from one display mode to another, the [UpdateHistogram](#) command should be used.

Syntax: [UpdateHistogram\(ImageHandle, Histogram_ID, mode_TOKEN,Result\)](#)

Parameters:

| | |
|-------------------------------|--|
| ImageHandle | The handle of a loaded bitmap or bitmap channel. This is only relevant for the RECALC display modes. Set to 0 otherwise. |
| Histogram_ID | The ID number returned by the Histogram command. |
| mode_TOKEN | One of the following display modes: |
| CUM NONCUM | Sets cumulative or non-cumulative mode. |
| RECALC_CUM | Recalculates the cumulative histogram before display. |
| RECALC_NONCUM | Recalculates the non-cumulative histogram before display. |
| Result | 1 if the command was successful, otherwise zero. |

Remarks:

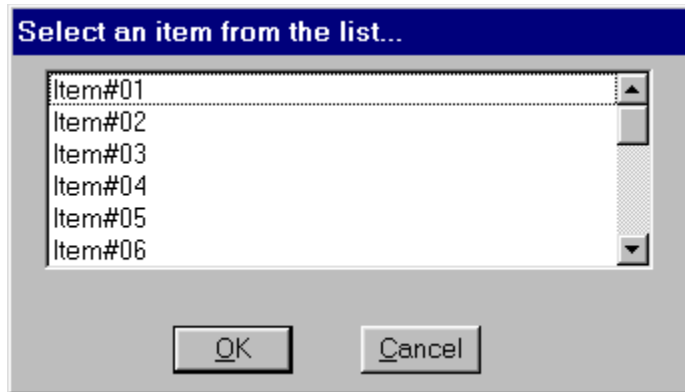
When a loaded bitmap brightness and contrast have been changed, for example, by the [NormalizeImage](#) or [EqualizeImage](#) command, the channel handles returned previously by [GetChannel](#) are no longer valid. Using the old values will result in the histogram not being updated. You have to make additional calls to [GetChannel](#) before the call to [UpdateHistogram](#).

Related Commands:

[Histogram](#) [GetChannel](#)

UpdateProgressBar

If a progress bar has been enabled, this command will update the display. The example image shows the effect. If a progress bar has not been enabled, this command has no effect.



Syntax: `UpdateProgressBar(Value,RELATIVE | ABSOLUTE | INCREMENT)`

Parameter:

Value

A number used to update the progress bar according to the TOKEN. These numbers must be in the range 0 - 100. This value is ignored if the **INCREMENT** mode is selected.

RELATIVE

The positive or negative integer value that the progress display is to be incremented. This number also sets the value that the **INCREMENT** mode will update the progress bar.

ABSOLUTE

The positive percentage value that is used to redraw the progress bar. For example, a value of 50 will draw the bar half completed.

INCREMENT

Updates the progress bar the amount last set with the **RELATIVE** mode.

Related Command:

[ProgressBar](#), [StatusWindow](#), [DrawStatusWinText](#)

UseBackground

This command has the dual role of controlling the background mode and the background color. The background mode establishes whether the GDI removes existing background colors before drawing any of the following:

- Text
- Shapes with a hatched brush
- Lines with a dotted pen

The GDI uses the background color to fill the small rectangles behind characters (called character cells), the gaps in dotted pens, and hatched lines in brushes.

Syntax: `UseBackground(OPAQUE/TRANSPARENT,r,g,b)`

Parameters:

OPAQUE Causes the background to be filled with the current color specified by *r, g, b*.

TRANSPARENT The background is left unchanged. This is the most commonly used.

r,g,b Specifies the color of the background using a combination of red, green, and blue. The default background color is white (255,255,255).

Remarks:

When you draw a character on the screen, the GDI does more than draw the squiggles that make up the character. It actually draws a rectangular area enclosing the character, called the *character cell*. The current font color determines the color of the characters, but the current background color controls the color in the character cells.

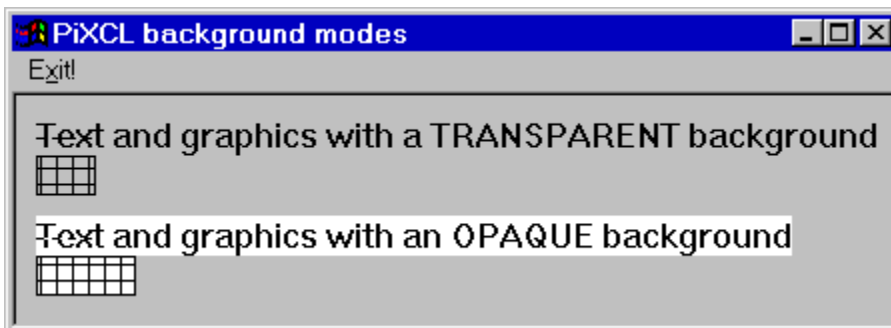
When the background mode is set to **OPAQUE**, the GDI fills the character cells with the RGB value you've set with the *r, g, b* parameters. On the other hand, when the background mode is set to **TRANSPARENT**, the GDI does not change the color in the character cells.

These same principles apply to hatched brushes and dotted pens. That is, when the background mode is set to **OPAQUE**, the GDI fills the gaps in dotted pens and hatched brushes with the RGB value you've set with the *r, g, b* parameters. When the background mode is set to **TRANSPARENT**, the GDI does not change the color in the gaps.

You can change the color of the entire window's background using the `DrawBackground` command. This command also erases any window contents. (The background mode setting--**OPAQUE** or **TRANSPARENT**--has no effect on the `DrawBackground` command.)

Example:

The following example shows the effect of the background mode and color settings when you draw text, draw lines with a dotted pen, and draw rectangles with a hatched brush. Notice that the white background only appears for character cells and the gaps in dotted pens and hatched brushes when the background mode is set to **OPAQUE**.



**The effect of the background mode and color settings.
The character cell is visible as the white background.**

```
{Change window's background to light gray}
    UseBackground(TRANSPARENT,192,192,192)
    DrawBackground

{Change background mode to TRANSPARENT and color to white}
    UseBackground(TRANSPARENT,255,255,255)

{Draw text, dotted line, and hatched rectangle}
    DrawText(10,10,"Text with a TRANSPARENT background")
    UsePen(DOT,1,0,0,0)      {Dotted pen}
    DrawLine(10,21,40,21)
    UsePen(SOLID,1,0,0,0)    {Reset the pen to solid}
    UseBrush(CROSS,0,0,0)    {Cross hatched brush}
    DrawRectangle(10,30,40,40)

{Change back to default background mode and color (white)}
    UseBackground(OPAQUE,255,255,255)

{Draw more text and another dotted line and hatched rectangle}
    DrawText(10,60,"Text with an OPAQUE background")
    UsePen(DOT,1,0,0,0)      {Dotted pen}
    DrawLine(10,71,40,71)
    UsePen(SOLID,1,0,0,0)    {Reset the pen to solid}
    UseBrush(CROSS,0,0,0)    {Cross hatched brush}
    DrawRectangle(10,80,40,90)

{Wait for input}
    WaitInput()
```

See [DrawRoundRectangle](#) for another example of UseBackground.

Related Commands:

[DrawBackground](#), [DrawText](#), [UsePen](#), [UseBrush](#), [DrawShadeRectangle](#) [GetBackground](#)

UseBrush

Defines the style and color of the brush that will be used in subsequent drawing operations.

Syntax:

```
UseBrush(SOLID/DIAGONALUP/DIAGONALDOWN/DIAGONALCROSS/  
HORIZONTAL/VERTICAL/CROSS/NULL,r,g,b)
```

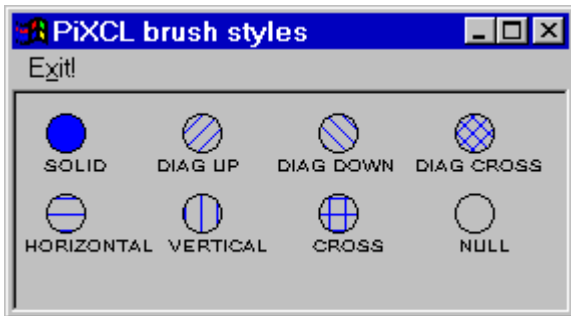
Parameters:

| | |
|----------------------|---|
| SOLID | A solid brush. |
| DIAGONALUP | 45-degree upward hatch (left to right). |
| DIAGONALDOWN | 45-degree downward hatch (left to right). |
| DIAGONALCROSS | 45-degree crosshatch. |
| HORIZONTAL | Horizontal hatch. |
| VERTICAL | Vertical hatch. |
| CROSS | Horizontal and vertical crosshatch. |
| NULL | A null or "hollow" brush (no color is drawn). |
| <i>r,g,b</i> | Specifies the color of the brush using a combination of red, green, and blue. |

Examples:

This program defines a crosshatched red brush and draws a round rectangle with it.

```
UseBrush (CROSS,255,0,0)  
DrawRoundRectangle (10,10,40,40,10,10)  
WaitInput ()
```



The eight different brush styles.

```
UseFont ("Arial",5,11,NOBOLD,  
NOITALIC,NONUNDERLINE,0,0,0)  
UseBrush (SOLID,0,0,255)  
DrawEllipse (15,10,35,30)  
DrawText (14,32,"SOLID")  
  
UseBrush (DIAGONALUP,0,0,255)  
DrawEllipse (83,10,103,30)
```

This next example draws a series of circles using the eight available brush styles. The program below produces the window at left.

```
DrawText (70, 32, "DIAGONALUP")
```

```
UseBrush (DIAGONALDOWN, 0, 0, 255)  
DrawEllipse (151, 10, 171, 30)  
DrawText (130, 32, "DIAGONALDOWN")
```

```
UseBrush (DIAGONALCROSS, 0, 0, 255)  
DrawEllipse (219, 10, 239, 30)  
DrawText (199, 32, "DIAGONALCROSS")
```

```
UseBrush (HORIZONTAL, 0, 0, 255)  
DrawEllipse (15, 50, 35, 70)  
DrawText (4, 72, "HORIZONTAL")
```

```
UseBrush (VERTICAL, 0, 0, 255)  
DrawEllipse (83, 50, 103, 70)  
DrawText (76, 72, "VERTICAL")
```

```
UseBrush (CROSS, 0, 0, 255)  
DrawEllipse (151, 50, 171, 70)  
DrawText (148, 72, "CROSS")
```

```
UseBrush (NULL, 0, 0, 255)  
DrawEllipse (219, 50, 239, 70)  
DrawText (221, 72, "NULL")
```

```
WaitInput()
```

Related Commands:

[DrawArc](#) [DrawChord](#) [DrawEllipse](#) [DrawFlood](#) [DrawPie](#) [DrawRectangle](#) [DrawRoundRectangle](#) [DrawTriangle](#)
[DrawPolygon](#) [UseBrushPattern](#)

UseBrushPattern

The [UseBrush](#) command provides solid and standard Windows hatch pattern fills. With the [UseBrushPattern](#) command, you can create your own fill pattern bitmaps in any of the supported bitmap formats. There are differences in operation between Windows 95/98 and NT 4.0.

In Windows 95/98: the image must be an 8x8 bitmap only. Microsoft does not support use of a larger bitmap, and the brush selection will fail. PiXCL sets the brush to the default.

In Windows NT 4.0: the image can be any size, and if possible the image will be tiled into the available fill region. Windows also produces a maximum size tile of about 256x256 from the selected image, even if it is larger.

Syntax: [UseBrushPattern\(GroupName\\$\)](#)

Parameters:

[ImageName\\$](#) The name of the pattern image file. If the image is not in memory, it is loaded. If [ImageName\\$](#) is null, the pattern brush flag is cleared. Changing [ImageName\\$](#) will change the brush bitmap, but does not clear the pattern brush flag.

Remarks:

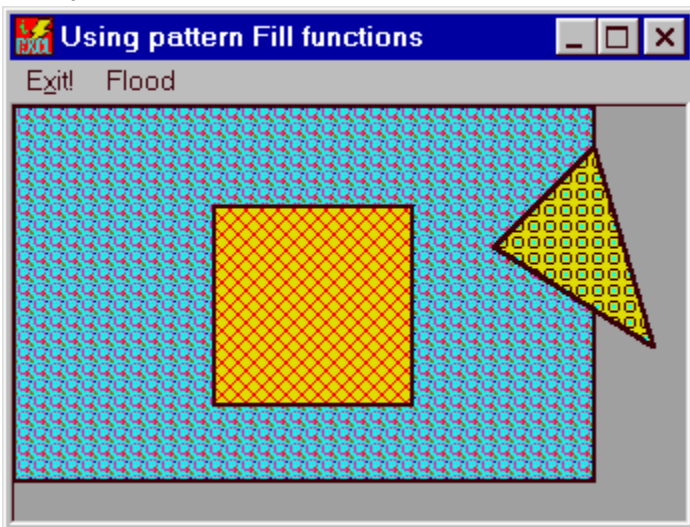
You can also load the pattern bitmap with the [LoadBitmap](#) command prior to using the pattern. Pattern bitmaps can be freed from memory with the [FreeBitmap](#) command.

The pattern bitmap is 8x8. We recommend drawing the bitmap using a 16 color palette (though you can use 256 or full color bitmaps as well), as the draw and fill operations will be faster. If you specify a larger bitmap, only the top left corner 8x8 section will be used. Pattern bitmaps can be created with any paint program such as MS-Paint that comes with Windows, or JASC™ Paint Shop Pro™ 4 or later.

You could also create your own pattern editor using PiXCL. For example, define a 64x64 (i.e. zoom factor 8) region, and draw with the mouse using flood commands. Save with the [SaveRectangle](#) command as a BMP file, then [ResizeImage](#) to make the 8x8 pattern bitmap desired.

The [UseBrush\(NULL,r,g,b\)](#) command does not clear a selected brush pattern image: place a [UseBrushPattern\(“”\)](#) in your script to do this.

Example:



This code fragment produces the window above, showing a combination of [UseBrush](#) and [UseBrushPattern](#) commands.

```
Bitmap_Color:  
  DrawBackground  
  LoadBitmap(Patt1$,FULL)  
  LoadBitmap(Patt2$,FULL)
```

```
LoadBitmap(Patt3$, FULL)
LoadBitmap(Patt4$, FULL)
WinGetClientRect("", cx1, cy1, cx2, cy2)
UsePen(SOLID, 2, 0, 0, 0)
UseBrush(NULL, 255, 255, 0)
DrawRectangle(cx1, cy1, cx2, cy2)
UseBrush(SOLID, 255, 255, 0)
DrawRectangle(100, 50, 200, 150)
UseBrush(DIAGONALCROSS, 255, 0, 0)
DrawRectangle(100, 50, 200, 150)
UseBrushPattern(Patt3$)
DrawTriangle(240, 70, 320, 120, 290, 20)
UseBrushPattern(Patt4$)
GetBackground(r, g, b)
DrawFloodExt(90, 50, r, g, b, SURFACE)
Goto Wait_for_Input
```

Related Commands:

[DrawArc](#), [DrawChord](#), [DrawEllipse](#), [DrawFlood](#), [DrawPie](#), [DrawRectangle](#), [DrawRoundRectangle](#), [DrawTriangle](#), [DrawPolygon](#), [LoadBitmap](#), [FreeBitmap](#), [UseBrush](#)

UseCaption

Updates a caption at the top of the PiXCL window, in the titlebar.

Syntax: [UseCaption\(Text\\$\)](#)

Parameter:

[Text\\$](#) The text you want to use for the caption.

Remark:

The [UseCaption](#) command is used to set the title of the current PiXCL application. You can also use the [WinTitle](#) command as a more powerful replacement for [UseCaption](#), because it lets you set the caption for any application window, including PiXCL's.

Example:

This command places the caption "Bluesky" at the top of the PiXCL window.

```
UseCaption("Bluesky")  
WaitInput()
```

Related Command:

[WinTitle](#)

UseCoordinates

Specifies PiXCL's coordinate system as either pixel or metric and controls the unit of measure that PiXCL uses for all subsequent drawing and window sizing operations.

Syntax: `UseCoordinates(PIXEL/METRIC)`

Parameters:

PIXEL Causes PiXCL to use pixels as the unit of measure.

METRIC Causes PiXCL to use millimeters as the unit of measure. METRIC is the default.

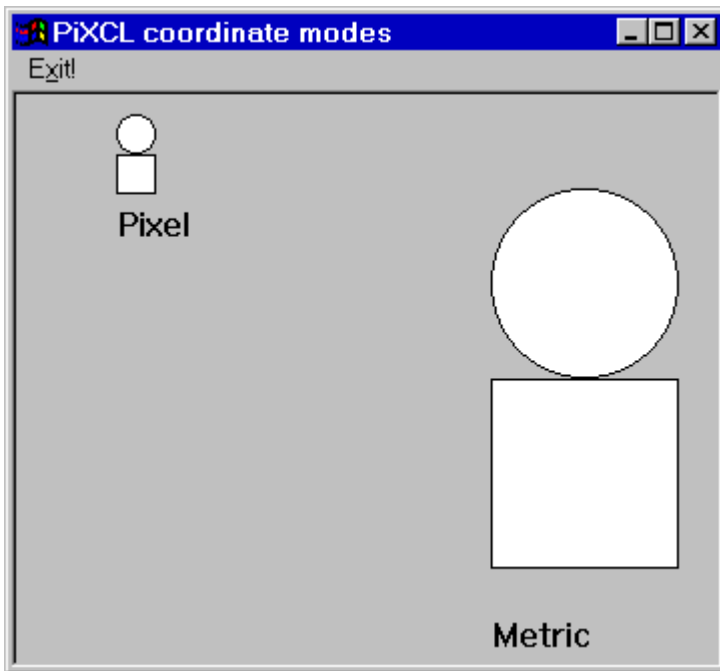
Remarks:

By using metric coordinates, you can make your programs more device independent. For example, any shapes you draw on the screen will appear the same regardless of the video display driver.

Programs written using pixel coordinates are more device dependent, which may or may not be a problem for you. For example, suppose you have a VGA system and you write a program that draws shapes in a window. When you run that program on a Super VGA system, the shapes will appear smaller. One virtue of pixel coordinates is that they are much more accurate than metric. Therefore, if you need to address the screen with the highest precision, you'll want to use pixel coordinates rather than metric.

One way to make your programs device independent without resorting to metric coordinates is to use the [GetScreenCaps](#) function to get the pixel resolution of the Windows display driver (use the HORZRES and VERTRES tokens). You can then perform your drawing and window resizing taking into account the resolution of the current display driver.

Example:



The following program draws the same shapes using pixel and metric coordinates. The different results are shown at left.

Pixel versus metric coordinates.

```
UseCoordinates (PIXEL)
DrawEllipse (50,10,70,30)
DrawRectangle (50,30,70,50)
```

```
DrawText (50, 55, "Pixel")
UseCoordinates (METRIC)
DrawEllipse (50, 10, 70, 30)
DrawRectangle (50, 30, 70, 50)
DrawText (50, 55, "Metric")
WaitInput ()
```

Related Commands:

All Draw commands, [GetScreenCaps](#)

UseCursor

Changes the mouse cursor to one of the predefined Windows cursors, or one of the cursors built into PiXCL.

Syntax:

```
UseCursor(APPSTARTING / ARROW / CROSS / IBEAM /  
          ICON / NO / SIZE / SIZEALL / SIZENESW /  
          SIZENS / SIZENWSE / SIZEWE / UPARROW /  
          WAIT / ZOOM / CROSSHAIR / TOPLEFT /  
          TOPRIGHT / BOTTOMLEFT / BOTTOMRIGHT)
```

Parameters:

| | |
|-------------|--|
| APPSTARTING | Standard arrow and small hourglass. |
| ARROW | Standard arrow. |
| CROSS | Crosshair. |
| IBEAM | Text I-beam. |
| ICON | Empty icon. |
| NO | Slashed circle. |
| SIZE | Four-pointed arrow. |
| SIZEALL | Same as SIZE. |
| SIZENESW | Double-pointed arrow pointing northeast and southwest. |
| SIZENS | Double-pointed arrow pointing north and south. |
| SIZENWSE | Double-pointed arrow pointing northwest and southeast. |
| SIZEWE | Double-pointed arrow pointing west and east. |
| UPARROW | Vertical arrow. |
| WAIT | Hourglass. |
| ZOOM | Magnifying glass. |
| CROSSHAIR | Crosshair for selecting a pixel. |
| TOPLEFT | Top left corner symbol. |
| TOPRIGHT | Top right corner symbol. |
| BOTTOMLEFT | Bottom left corner symbol. |
| BOTTOMRIGHT | Bottom right corner symbol. |

Remarks:

The [UseCursor](#) command has an effect only under these circumstances:

- The new cursor is different from the previous cursor.
- PiXCL is waiting for mouse input; a [SetMouse](#), [SetCtrlMouse](#), [SetRightMouse](#), [SetShftRightMouse](#) or [SetDbI Mouse](#) is in effect.

Example:

The following example changes the mouse cursor to a crosshair. It then draws a label at the current cursor location when the user clicks the mouse.

```
{Get screen capacity in pixels}  
  UseCoordinates(PIXEL)  
  GetScreenCaps(HORZRES, PixelsX)  
  GetScreenCaps(VERTRES, PixelsY)  
  
{Use a cross-hair cursor}  
  UseCursor(CROSS)
```

```
{Set up mouse hit-testing}
  SetMouse (0,0, PixelsX, PixelsY, LabelIt, MouseX, MouseY)

{Wait for input}
Wait_for_input:
  WaitInput()

{Draw the label "Click!" whenever the mouse is clicked}
LabelIt:
  DrawText (MouseX, MouseY, "Click!")
  Goto Wait_for_input
```

Related Commands:

[SetCtrlMouse](#), [SetMouse](#), [SetDbIMouse](#), [SetRightMouse](#), [WaitInput](#)

UseFont

Establishes the font that will be used for subsequent text-drawing operations in PiXCL. It lets you specify the width, height, style (bold, italics, and underlining), and color of the font.

Syntax:

```
UseFont(Name$,Width,Height,Set_bold,Set_italic,  
        Set_underline,r,g,b)
```

Parameters:

| | |
|----------------------|---|
| <i>Name\$</i> | The name of the font. |
| <i>Width</i> | The width of the font using the current coordinate mode (either pixels or millimeters). If you specify a width of zero, then the default width is used for the given font. |
| <i>Height</i> | The height of the font using the current coordinate mode (either pixels or millimeters). If you specify a height of zero, then the default height is used for the given font. |
| <i>Set_bold</i> | Controls whether the font is bold and must be one of the following tokens: |
| BOLD | Font is bold |
| NOBOLD | Font is not bold |
| <i>Set_italic</i> | Controls whether the font is italic and must be one of the following tokens: |
| ITALIC | Font is italic |
| NOITALIC | Font is not italic |
| <i>Set_underline</i> | Controls whether the font is underlined and must be one of the following tokens: |
| UNDERLINE | Font is underlined |
| NOUNDERLINE | Font is not underlined |
| <i>r,g,b</i> | The color of the font using a combination of red, green, and blue. |

Remarks:

The GDI maintains all the fonts that are available in the system in a font table. When you describe a font with the [UseFont](#) command, it may or may not exist in the GDI's table. In a process known as *font mapping*, the GDI compares the font parameters you've supplied--point size, serif or sans serif, and fixed or proportional--to the fonts it has in its table and uses the font that is the closest match.

In some cases, the GDI will synthesize a font to match the parameters you've supplied. In early versions of Windows, when the GDI synthesized a font, it would sometimes match only the point size. In Windows NT and Windows 95, because TrueType fonts are used, font mapping is more reliable. In fact, if you specify a TrueType font for *Name\$*, it will always display the attributes you request.

To see a list of the fonts available in your system, activate the Fonts icon in Control Panel. To use one of these fonts in an PiXCL program, simply supply its name as the *Name\$* parameter in UseFont. Just be sure to spell the name exactly as you see it on the screen. The standard fonts that are available in Windows NT are MS Sans Serif, MS Serif, Courier, Small Fonts, Modern, Roman, Script, Terminal, Arial, Times New Roman, Courier New, Symbol, System, and Wingdings (see the second example).

Certain TrueType fonts already have bold and italic attributes--for example, Times New Roman Bold Italic. When using these fonts, the tokens you specify for *Set_bold*, *Set_italic*, and *Set_underline* have no effect. Nevertheless, as long as you provide the full name of the font in the *Name\$* parameter, PiXCL will use the font, complete with attributes, without a problem.

PiXCL also lets you access fonts you've added with third-party font packages. The names of these fonts may or may not be listed in Control Panel.

The default font is the System font in black.

If the color that appears behind each character is not what you want, you may need to change the background mode or color. See the [UseBackground](#) command for more details.

Examples:

The following program shows the effect of explicitly controlling the width and height of a font versus using the default width and height.

```
{Set the coordinate mode to pixel}
  UseCoordinates(PIXEL)

{Set font's width to 10 and height to 20 and draw text}
  UseFont("Times New Roman",10,20,
  NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
  DrawText(10,10,
  "Times New Roman with specified height and width")

{Use the default width and height to draw text}
  UseFont("Times New Roman",0,0,
  NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
  DrawText(10,40,
  "Times New Roman with default height and width")

  WaitInput()
```

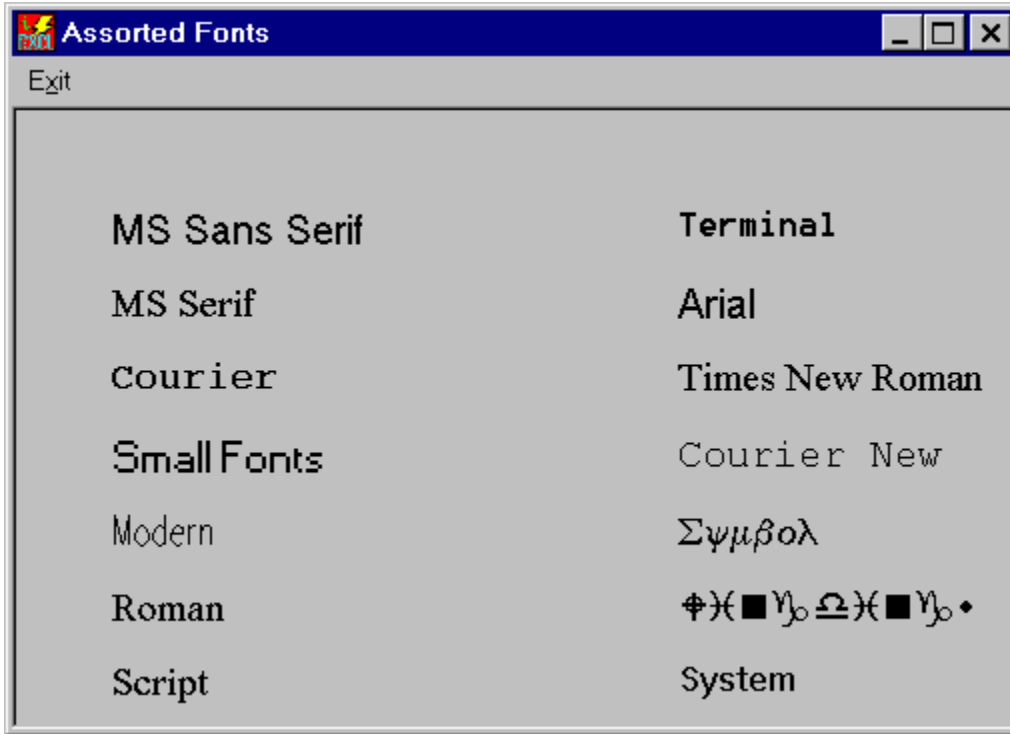
This next program shows some samples of the standard fonts available in Windows 95 and NT. The standard fonts are MS Sans Serif, MS Serif, Courier, Small Fonts, Modern, Roman, Script, Terminal, Arial, Times New Roman, Courier New, Symbol, Wingdings, and System.

```
UseCoordinates(METRIC)
UseFont("MS Sans Serif",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,10,"MS Sans Serif")
UseFont("MS Serif",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,18,"MS Serif")
UseFont("Courier",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,26,"Courier")
UseFont("Small Fonts",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,34,"Small Fonts")
UseFont("Modern",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,42,"Modern")
UseFont("Roman",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,50,"Roman")
UseFont("Script",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(10,58,"Script")
UseFont("Terminal",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(70,10,"Terminal")
UseFont("Arial",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(70,18,"Arial")
UseFont("Times New Roman",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(70,26,"Times New Roman")
UseFont("Courier New",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(70,34,"Courier New")
UseFont("Symbol",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText(70,42,"Symbol")
UseFont("Wingdings",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
```

```
DrawText (70,50,"Wingdings")
UseFont ("System",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
DrawText (70,58,"System")
```

```
WaitInput ()
```

The above code generates the window below.



Related Commands:

[AddFont](#), [RemoveFont](#), [GetTextSpacing](#), [SetTextSpacing](#), [DrawText](#), [DrawTextExt](#), [UseBackground](#), [ChooseFont](#), [UseFontExt](#)

UseFontExt

This command is designed to be compatible with the [ChooseFont](#) command that uses the COMDLG32.DLL library function. The main difference between [UseFontExt](#) and [UseFont](#) is that there are integer variables, [Bold](#), [Italic](#) and [Underline](#) in place of the [\(NO\)BOLD](#), [\(NO\)ITALIC](#) and [\(NO\)UNDERLINE](#) tokens, plus a new variable [Strikeout](#).

Syntax: [UseFontExt](#)(*Font\$,Width,Height,Bold,Italic,Underline,Strikeout,r,g,b*)

Parameters:

| | |
|---|---|
| Font\$ | The name of the font. |
| Width,Height | The width of the font using the current coordinate mode (either pixels or millimeters). If you specify a width of zero, then the default width is used for the given font. |
| Bold,Italic,Underline,Strikeout | The Bold variable takes a value of either NORMAL (400), or BOLD (700), and the Italic, Underline, Strikeout variables take values of either 0 (e.g. NOBOLD), or 1 (BOLD). If these variables are assigned any value other than 0, they are internally converted to 1 before being used. |

Related Commands

[ChooseFont](#), [DrawText](#), [DrawTextExt](#), [UseFont](#)

UsePen

Establishes the pen that will be used for subsequent drawing operations. It lets you specify the style, width, and color for the pen.

Syntax: `UsePen(Style,Width,r,g,b)`

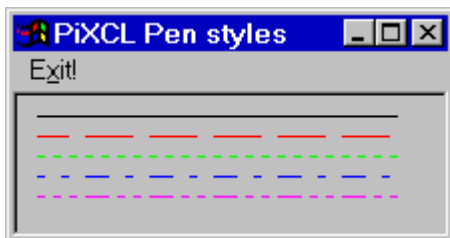
Parameters:

| | |
|--------------|---|
| <i>Style</i> | The style of the pen. It must be one of the tokens in the table below. The default is SOLID. |
| <i>Width</i> | The width of the line in pixels. It must be 1, unless the pen style is SOLID or NULL. The default width is 1. |
| <i>r,g,b</i> | Specifies the color of the pen using a combination of red, green, and blue. The default is black (0,0,0). |

| Token | Description | Result |
|--------------|--------------------|---------------|
| SOLID | Solid line | _____ |
| DASH | Dashed line | - - - - - |
| DOT | Dotted line | |
| DASHDOT | Dash-dot line | - . - . - . |
| DASHDOTDOT | Dash-dot-dot line | - . . - . . |
| NULL | No line | |

Table: **Pen Styles**

Example:



The following example draws five lines, each using a different pen style and color, and produces the windows at left.

The effect of different pens.

```
{Solid line in black}
UsePen(SOLID,1,0,0,0)
DrawLine(10,10,90,10)
```

```
{Dashed line in red}
UsePen(DASH,1,255,0,0)
DrawLine(10,20,90,20)
```

```
{Dotted line in green}
UsePen(DOT,1,0,255,0)
DrawLine(10,30,90,30)
```

```
{Dash-dot line in blue}  
UsePen(DASHDOT,1,0,0,255)  
DrawLine(10,40,90,40)
```

```
{Dash-dot-dot line in pink}  
UsePen(DASHDOTDOT,1,255,0,255)  
DrawLine(10,50,90,50)  
WaitInput()
```

Related Commands:

All the Draw commands that draw lines and shapes.

Val

Converts a string to a number.

Syntax: *Val(String\$,Number,Result)*

Parameters:

| | |
|-----------------|---|
| <i>String\$</i> | A string representing the number you want to convert. |
| <i>Number</i> | An integer variable that will contain the converted number. |
| <i>Result</i> | An integer variable that indicates the outcome of the conversion. If it was successful, this variable is assigned a value of 1. If it was not successful, this variable is assigned a value of 0. |

Remarks:

The numeric string you provide in *String\$* must represent an integer from -2147483647 to 2147483648.

If a positive number exceeds 10 digits or a negative number exceeds 11 digits, this function returns with *Number* = 0 and *Result* = 0.

Examples:

This example uses the [TextBox](#) command to prompt you for a number in the form of a string. It then converts the numeric string to an integer using Val.

Again:

```
Number$ = "1"
TextBox("Enter a number",
        "Val function test",Number$,Button)
If Button = 2 Then End           {Cancel selected}
Val(Number$,Number,Result)
If Result = 1 Then Goto OK
MessageBox(OK,1,EXCLAMATION,"Invalid number","Invalid",Button)
Goto Again:
```

OK:

```
TextOut$ = "You entered " + Number$
MessageBox(OK,1,EXCLAMATION,TextOut$,"Number entered",Button)
```

Related Command:

[Str](#) [FpVal](#)

Val64

PIXCL 5.1 command. Converts a string to a 64-bit number.

Syntax: *Val64(String\$,Number64#,Result)*

Parameters:

String\$ A string representing the number you want to convert.

Number64# A 64-bit integer variable that will contain the converted number.

Result An integer variable that indicates the outcome of the conversion. If it was successful, this variable is assigned a value of 1. If it was not successful, this variable is assigned a value of 0.

Related Command:

[Str64](#)

VidAutoPalette

PIXCL 5 Command. Assuming the selected device driver supports palettes, you can generate a video palette. This is only relevant when capturing 8 bits per pixel. Most new video frame devices (e.g. USB) support 24 bits per pixel.

Syntax: *VidAutoPalette(VideoHandle, SampleFrames, MaxColours, Result)*

Parameters:

| | |
|---------------------|--|
| <i>VideoHandle</i> | A non zero handle from <i>VidCreateCaptureWindow</i> . |
| <i>SampleFrames</i> | The number of frames to sample to make the palette. |
| <i>MaxColours</i> | A number between 1 and 256. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

VidLoadPalette VidSavePalette

VidCamConnect

PIXCL5 Command. Select an available video capture device. If your camera is Video-For-Windows or DirectShow compatible, this command should work.

Syntax: *VidCamConnect(VideoHandle,Index,Result)*

Parameters:

VideoHandle A non zero handle from [VidCreateCaptureWindow](#).
Index A driver index value of 0 – 9.
Result 1 if the operation was successful, otherwise 0.

Related Commands:

[VidCamDisconnect](#)

VidCamDisconnect

PIXCL5 Command. Deselect an available video capture device. If your camera is Video-For-Windows or DirectShow compatible, this command should work.

Syntax: [VidCamDisconnect\(VideoHandle,Index,Result\)](#)

Parameters:

[VideoHandle](#) A non-zero handle from [VidCreateCaptureWindow](#).
[Index](#) A driver index value of 0 – 9.
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[VidCamConnect](#)

VidCaptureDlg

PIXCL 5 Command. Video sources have four possible dialogs that set up the source parameters. Note that the dialogs (especially FORMAT) displayed are part of the video driver files installed with your camera. If you have multiple cameras, you will see different dialogs depending on which camera is selected.

Syntax: [VidCaptureDlg\(VideoHandle,COMPRESSION,Result\)](#)

Parameters:

| | |
|-----------------------------|--|
| VideoHandle | A non zero handle from VidCreateCaptureWindow . |
| Token | COMPRESSION, DISPLAY, FORMAT, SOURCE. |
| Result | 1 if the dialog is displayed, otherwise 0. Note that the button pushed does not return any variable. |

Related Commands:

[VidCreateCaptureWindow](#)

VidCaptureSingleFrame

PIXCL 5 Command. Add a single frame to the capture file specified in [VidSetCaptureFile](#).

Syntax: [VidCaptureSingleFrame](#)(*VideoHandle*,*Result*)

Parameters:

[VideoHandle](#) A non zero handle from [VidCreateCaptureWindow](#).

[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[VidCreateCaptureWindow](#) [VidSetCaptureFile](#)

VidCaptureSingleFrameClose

PIXCL 5 Command. Close the capture file specified in [VidSetCaptureFile](#).

Syntax: [VidCaptureSingleFrameClose](#)(*VideoHandle*,*Result*)

Parameters:

[VideoHandle](#) A non zero handle from [VidCreateCaptureWindow](#).

[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[VidCreateCaptureWindow](#) [VidSetCaptureFile](#)

VidCaptureSingleFrameOpen

PIXCL 5 Command. Open the capture file specified in [VidSetCaptureFile](#).

Syntax: [VidCaptureSingleFrameOpen](#)(*VideoHandle*,*Result*)

Parameters:

[VideoHandle](#) A non zero handle from [VidCreateCaptureWindow](#).

[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[VidCreateCaptureWindow](#) [VidSetCaptureFile](#)

VidClipboardPutBitmap

PIXCL 5 Command. Copy the current frame to the clipboard as a BMP file. The [ClipboardGetBitmap](#) command is usually then used to copy the bitmap into the PIXCL image list, where you can perform any of the PIXCL image processing and handling functions.

Syntax: [VidClipboardPutBitmap](#)(*VideoHandle*,*Result*)

Parameters:

[VideoHandle](#) A non zero handle from [VidCreateCaptureWindow](#).
[Result](#) 1 if the operation was successful, otherwise 0.

Related Commands:

[VidCreateCaptureWindow](#) [VidSetCaptureFile](#) [ClipboardGetBitmap](#)

VidCloseCaptureWindow

PIXCL 5 Command. Use this command to close any of the open video capture windows. Its preferable that you use a [VidCamDisconnect](#) command prior to this command, especially if you have more than one camera on your system.

Syntax: [VidCloseCaptureWindow\(VideoHandle\)](#)

Parameters:

[VideoHandle](#) A handle returned by the [VidCaptureWindow](#) command.

Related Commands:

[VidCamDisconnect](#) [VidCaptureWindow](#)

VidCaptureWindow

PIXCL 5 Command. This command, and the rest of the **Vid*** command set work with Video-For-Windows or DirectShow compatible devices, and create a child window within the PiXCL client area that displays the available video stream. You can make up to 10 windows with a different camera displayed on each one. Note that Preview windows take a LOT of cpu time, so don't expect that 6 windows will have smooth video. Perhaps when cpu clocks reach 2GHz ...

Syntax: `VidCaptureWindow(Title$,x1,y1,x2,y2,Index, FrameRate,PreviewMode,VideoHandle)`

Parameters:

| | |
|--------------------|--|
| <i>Title\$</i> | The title to appear in the title bar of the window, which makes the window movable. If an empty string, no title bar is displayed, and the window is not movable. |
| <i>x1,y1,x2,y2</i> | The client area coordinates for the video window. |
| <i>Index</i> | The camera driver index 0 – 9. The default is 0. See the VidGetDriverDescription command. |
| <i>FrameRate</i> | The number of frames per second to be displayed. 15 is often the supported rate. In general the video hardware driver will set the closest rate specified. |
| <i>PreviewMode</i> | 0 or @TRUE to enable preview mode i.e. live video feed to the Window. 1 or @FALSE to set the mode to capture the first available frame. The mode is bit 0 in the integer value. Bit 1 = 1 creates a status window on the video display and writes the frame number. Bit 2 = 1 sets a buffer that averages the current and previous frame. This can be helpful to clean up quantization jitter, if speed is not an issue. |
| <i>VideoHandle</i> | A non-zero handle of the created window, used with most of the other Vid* commands. |

Related Commands:

[VidGetDriverDescription](#) [VidSetCaptureFile](#)

VidEnablePreview

PIXCL 5 Command. The video preview mode and frame rate can be set with this command.

Syntax: `VidEnablePreview(VideoHandle, FrameRate, PreviewMode)`

Parameters:

VideoHandle A non-zero handle from `VidCreateCaptureWindow`.

FrameRate The number of frames per second to be displayed. 15 is often the supported rate. In general the video hardware driver will set the closest rate specified.

PreviewMode **@TRUE** to enable preview mode i.e. live video feed to the Window.
@FALSE to set the mode to capture the first available frame.

Related Commands:

[VidCreateCaptureWindow](#)

VidGetCaptureFile

PIXCL 5 Command. The video capture set with this command. The default is `c:\capture.avi`.

Syntax: `VidGetCaptureFile(VideoHandle,Filename$)`

Parameters:

VideoHandle A non zero handle from `VidCreateCaptureWindow`.

FileName\$ The capture filename.

Related Commands:

[VidCreateCaptureWindow](#)

VidGetDevCaps

PIXCL 5 Command. The current video device capabilities are obtained with this command.

Syntax: `VidGetDevCaps(VideoHandle,DriverName$,CapsArray[Start],Result)`

Parameters:

| | |
|---------------------|---|
| <i>VideoHandle</i> | A non zero handle from VidCreateCaptureWindow . |
| <i>DriverName\$</i> | The current video capture drivename reported. |
| <i>CapsArray[]</i> | A previously defined integer array of at least 8 elements. In order, these capabilities are: [0] = DeviceIndex: 0 - 9 [1] = HasVideoOverlay: 0, or 1 if the device supports video overlay. [2] = HasVideoSourceDlg: 0 or 1 [3] = HasVideoFormatDlg: 0 or 1 [4] = HasVideoDisplayDlg: 0 or 1 [5] = CaptureInitialized: 0 or 1 [6] = DvrSuppliesPalettes: 0 or 1 |
| <i>Start</i> | The start element index in the array. This will usually be 0. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Commands:

[VidCreateCaptureWindow](#)

VidGetDriverDescription

PIXCL 5 Command. The installed video capture drivers can be listed with this command.

Syntax: [VidGetDriverDescription\(Index, NameVersion\\$\)](#)

Parameters:

[Index](#) Installed drivers have an index of 0 – 9.

[NameVersion\\$](#) The returned name and version string.

Related Commands:

[VidCamConnect](#) [VidCamDisconnect](#)

VidGetStatus

PIXCL 5 Command. The video device status is returned by this command. Just two values are returned at present, but there are many more that may be added in the future.

Syntax: `VidGetStatus(VideoHandle, VideoWidth, VideoHeight)`

Parameters:

VideoHandle A non zero handle from [VidCreateCaptureWindow](#).
VideoWidth The width in pixels of the current video mode.
VideoHeight The height in pixels of the current video mode.

Related Commands:

[VidCreateCaptureWindow](#)

VidLoadPalette

PIXCL 5 Command. The video capture device driver may support loading a palette. This can be useful where you need to colourize a monochrome video stream.

Syntax: `VidLoadPalette(VideoHandle, VideoPaletteFile$, Result)`

Parameters:

VideoHandle A non zero handle from [VidCreateCaptureWindow](#).
VideoPaletteFile\$ The name of the palette file. We recommend calling these *.vpl to not confuse them with .pal files.
Result 1 if the operation was successful, otherwise 0.

Related Commands:

[VidSavePalette](#)

VidOverlay

PIXCL 5 Command. The video capture device driver may support overlays (often in hardware). This command enables or disables overlays for the selected device.

Syntax: `VidOverlay(VideoHandle, @TRUE | @FALSE)`

Parameters:

| | |
|--------------------------|--|
| <code>VideoHandle</code> | A non zero handle from <code>VidCreateCaptureWindow</code> . |
| <code>@TRUE</code> | Enable overlay. |
| <code>@FALSE</code> | Disable overlay. |

Related Commands:

[VidLoadPalette](#)

VidPreviewScale

PIXCL 5 Command. The video capture device driver supports scaling the video stream to the video preview window.

Syntax: `VidPreviewScale(VideoHandle, @TRUE | @FALSE)`

Parameters:

| | |
|---------------------------------|--|
| <code><i>VideoHandle</i></code> | A non zero handle from <code>VidCreateCaptureWindow</code> . |
| <code>@TRUE</code> | Enable scaling. |
| <code>@FALSE</code> | Disable scaling. |

Related Commands:

[VidEnablePreview](#)

VidSavePalette

PIXCL 5 Command. The video capture device driver may support saving a palette. This can be useful where you need to colourize a monochrome video stream.

Syntax: `VidSavePalette(VideoHandle, VideoPaletteFile$, Result)`

Parameters:

VideoHandle A non zero handle from [VidCreateCaptureWindow](#).
VideoPaletteFile\$ The name of the palette file. We recommend calling these *.vpl to not confuse them with .pal files.
Result 1 if the operation was successful, otherwise 0.

Related Commands:

[VidLoadPalette](#)

VidSetCaptureFile

PIXCL 5 Command. The video capture file is obtained with this command. The default is **c:\capture.avi**.

Syntax: `VidSetCaptureFile(VideoHandle,Filename$)`

Parameters:

VideoHandle A non zero handle from `VidCreateCaptureWindow`.

FileName\$ The capture filename.

Related Commands:

[VidCreateCaptureWindow](#)

ViewFilterFile

A 5x5 or 15x15 filter file can be displayed in a MessageBox with this command.

Syntax: [ViewFilterFile\(FilterFile\\$\)](#)

Parameter:

[FilterFile\\$](#) The filter file to read and display in a MessageBox. If the file cannot be read, or does not exist, the command is ignored.

Related Files:

[Create5x5Filter](#)

WaitCommEvent

This command is issued once you are ready to send or receive data via a serial port. It should be followed by a [WaitInput\(\)](#) command.

Syntax: [WaitCommEvent\(R|W,Label,Timeout\)](#)

Parameters:

[R](#) | [W](#)

[R](#): wait for a CR-LF terminator to be received.

[W](#): wait till the last character in the buffer is written. If the serial device written has to do some processing, this may take a few seconds.

[Label](#)

The jump-to location in your script to process the event.

[Timeout](#)

The timelimit after which an event is issued anyway. This may happen if the attached device is not responding, so your program will have to be able to handle this situation. For example, if a Read timeout occurs, the buffer will be empty, and [ReadCommPort](#) will return a NULL string. [Timeout](#) has a default value of 5000 mS (i.e. 5 minutes). If you set [Timeout](#) to <= 0, it defaults to 5000.

Remarks:

[WaitCommEvent](#) will interact with the [WaitInput\(\)](#) idle loop. If a [WaitCommEvent](#) is active, and a mouse or keyboard event occurs, this will be processed immediately. You can set a flag variable to indicate that a [WaitCommEvent](#) is active, and put a [WaitCommEvent](#) into a temporary idle loop. Please remember that you can't put a [WaitInput\(\)](#) within a subroutine, because the keyboard and mouse events are disabled until the [Return](#) is processed.

Related Commands:

[ClearCommPort](#) [EscCommFunction](#) [GetCommPort](#) [ReadCommPort](#) [SetCommPort](#) [WriteCommPort](#)

WaitInput

Pauses a program a specified number of milliseconds or pauses it indefinitely to wait for user input. This can be input either for the PiXCL application, or to also let other applications operate.

Syntax:

`WaitInput()`

or

`WaitInput(Milliseconds)`

Parameter:

Milliseconds The number of milliseconds you want to pause the program.

Remarks:

`WaitInput` without an argument pauses a program indefinitely to wait for user input. When the user presses a key, clicks the mouse, or makes a menu selection and a `Button`, `SetKeyboard`, any `SetMouse`, or `SetMenu` command is in effect, PiXCL immediately transfers control to the appropriate label specified in one of these commands. For example, if a `SetMouse` command is in effect and the user clicks the mouse within a specified hit-testing region, control transfers to the label associated with that region. You can also send a PiXCL application a resume command from another PiXCL application with the `PXLResume` and `PXLResumeAt` commands. Note that the script will continue execution immediately after the `WaitInput()` command for the `PXLResume` command.

`WaitInput` with an argument pauses a program a specified number of milliseconds. For example, `WaitInput(1000)` pauses PiXCL for 1 second, `WaitInput(3000)` for three seconds, and `WaitInput(250)` for a quarter second.

You will also find that `WaitInput(100)` is useful for allowing Windows to update toolbar buttons, push buttons and status bars before the script continues processing. For example, when you click a toolbar button, script execution will jump to the defined **label**, but Windows needs to be given time to redraw the button from pressed to unpressed mode, otherwise the button will appear blank until your program gives Windows some processing time.

Because the PC's clock ticks only 16 times a second, the granularity of the `WaitInput` command is not as fine as a *milliseconds* argument might have you believe. For example, `WaitInput(1)` has the same effect as `WaitInput(62)`; they both pause your program for approximately 1/16 of a second.

`WaitInput(0)` is treated the same as `WaitInput(1)`.

Please note that these wait periods are approximate, and cannot and should not be used in any real time application that relies on exact timing.

As a general rule, you should always use `WaitInput()` when you are not performing work in the PiXCL window. This gives more of the system's resources to other programs at a time when your PiXCL program does not need them.

Because of Windows 95 and NT's multitasking architecture, you can pause PiXCL after executing a `Run` command, using some sort of polling operation. See also the `PXLResume` and `PXLResumeAt` commands.

Example:

The following program displays a message on the screen, pauses for 2 seconds, places another message on the top of the previous one, and then pauses indefinitely until you close the window.

```
DrawText(10,10,"PiXCL will pause for 2 seconds")
WaitInput(2000)
DrawText(10,10,
  "PiXCL will now pause indefinitely until you close the window") WaitInput()
```

See the [SetMenu](#), [SetMouse](#), and [SetKeyboard](#) commands for other examples of [WaitInput](#).

Remarks:

If you are writing additional programs with a C or C++ compiler, the resume message can be declared as

```
#define WM_USER_INTERPRETSCRIPT      (WM_USER+50)
#define WM_USER_INTERPRETSCRIPTAT(WM_USER+53)
```

Related Commands:

[SetWaitMode](#), [SetMenu](#), [SetKeyboard](#), any [SetMouse](#), [PXLResume](#) and [PXLResumeAt](#)

WAVGetDevCaps

If a WAV sound file play device (e.g. A SoundBlaster™ compatible) is installed, you can query the device to find out the capabilities of the board.

Syntax: [WAVGetDevCaps\(Device, TOKEN, ReturnString\\$, Result\)](#)

Parameters:

| | |
|--------------------------------|---|
| Device | The 0 indexed device number. If only one sound card is fitted, this is always zero. |
| PLAYERNAME | A string stored in the registry when the board is installed. |
| FORMATS | The best capability of the card, see remarks below. |
| NUMCHANNELS | Returns MONO or STEREO. |
| FUNCTIONS | Returns the modifiable available functions e.g. Pitch, PlayRate, Volume. |
| ReturnString\$ | The device cap string. |

[Result](#) 0 if the function fails, otherwise 1.

Remarks:

The possible WaveForm Device Formats are

"11.025 kHz, mono, 8-bit"
"11.025 kHz, mono, 16-bit"
"11.025 kHz, stereo, 8-bit"
"11.025 kHz, stereo, 16-bit"
"22.05 kHz, mono, 8-bit"
"22.05 kHz, mono, 16-bit"
"22.05 kHz, stereo, 8-bit"
"22.05 kHz, stereo, 16-bit"
"44.1 kHz, mono, 8-bit"
"44.1 kHz, mono, 16-bit"
"44.1 kHz, stereo, 8-bit"
"44.1 kHz, stereo, 16-bit"

If your card reports, say, "22.05 kHz, stereo, 16-bit", then it will support all the preceding formats.

Related Commands:

[WAVGetNumDevs](#) [WAVGetPitch](#) [WAVGetPlayRate](#) [WAVPlaySound](#) [WAVSetPitch](#) [WAVSetPlayRate](#)

WAVGetNumDevs

This command reports the number of WAV output devices installed in your computer.

Syntax: [WAVGetNumDevs\(*Number*\)](#)

Parameters:

Number

If only one sound card is fitted, this will return a value of 1. Note that sound device numbers are 0 indexed, hence most computers with one sound card will always use device number 0.

Related Commands:

[WAVGetDevCaps](#) [WAVGetPitch](#) [WAVGetPlayRate](#) [WAVPlaySound](#) [WAVSetPitch](#) [WAVSetPlayRate](#)

WAVGetPitch

Sound cards such as the industry standard SoundBlaster (TM Creative Labs) may support playback pitch (i.e. the sound playback frequency) changes. This command returns the current pitch setting as a percentage. The default value is 100.

Syntax: [WAVGetPitch\(Device,Pitch\)](#)

Parameters:

[Device](#) The device number of the sound device. For a system with one sound card, this will be 0.

[Pitch](#) The default value is 100, and the WAV file will play as recorded.

Remarks:

Not all sound cards support Pitch control. Use the [WAVGetDevCaps](#) command to list the available controls for your card.

Related Commands:

[WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPlayRate](#) [WAVPlaySound](#) [WAVSetPitch](#) [WAVSetPlayRate](#)

WAVGetPlayRate

Sound cards such as the industry standard SoundBlaster (TM Creative Labs) may support playback rate control. This command returns the current rate setting as a percentage. The default value is 100.

Syntax: [WAVGetPlayRate\(Device,PlayRate\)](#)

Parameters:

[Device](#) The device number of the sound device. For a system with one sound card, this will be 0.

[PlayRate](#) The default value is 100, and the WAV file will play as recorded.

Remarks:

Not all sound cards support playback rate control. Use the [WAVGetDevCaps](#) command to list the available controls for your card.

Related Commands:

[WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPitch](#) [WAVPlaySound](#) [WAVSetPitch](#) [WAVSetPlayRate](#)

WAVGetVolume

Sound cards such as the industry standard SoundBlaster (TM Creative Labs) may support volume control. This command returns the current volume setting as a percentage. The default value is 50.

Syntax: [WAVGetVolume\(Device,LeftVol, RightVol,Result\)](#)

Parameters:

[Device](#) The device number of the sound device. For a system with one sound card, this will be 0.

[LeftVol, RightVol](#) The default value is 50, and the WAV file will play as recorded.

[Result](#) 1 if the operation was successful. If the operation fails, the value may be
0 = could not open the play device.
-1 = INVALID DEVICE specified
-2 = NO DRIVER loaded
-3 = NO MEMORY for operation
-4 = function NOT SUPPORTED

Remarks:

Not all sound cards support volume control. Use the [WAVGetDevCaps](#) command to list the available controls for your card.

Related Commands:

[WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPitch](#) [WAVPlaySound](#) [WAVSetPitch](#) [WAVSetPlayRate](#)

WAVPlaySound

This function plays a specified waveform audio (.WAV) file or an entry in the [sounds] section of the registry. In Windows 3.1 and NT 3.51, there is a set of WAV files in the windows directory. These are

[chimes.wav](#) [chord.wav](#) [ding.wav](#) [mmsound.wav](#) [tada.wav](#)

In Windows 95 and NT 4 the so called "Microsoft Sound" is in [c:\windows\media\themec~1.wav](#). There are numerous WAV files that can be found by looking around the Internet and on bulletin boards.

Syntax:

`WAVPlaySound(Sound,$,SYNC/ASYNC/LOOP,ALIAS/FILENAME,Result)`

Parameters:

| | |
|--------------------------|---|
| Sound \$ | A waveform sound filename or the name of an entry in the [sounds] section of the registry. If this parameter is a null string (" "), any currently playing sound is stopped. |
| SYNC | Causes PiXCL to play the sound synchronously, and the WAVPlaySound function does not return until the sound ends. |
| ASYNC | Causes PiXCL to play the sound asynchronously, and the function returns immediately after beginning the sound. To terminate an asynchronously played sound, WAVPlaySound must be called again with Sound \$ set to a NULL string. |
| LOOP | The sound plays asynchronously and repeatedly until WAVPlaySound is called again with the Sound \$ parameter set to NULL. |
| ALIAS | The Sound \$ parameter is a system-event alias in the registry or the WIN.INI file. |
| FILENAME | The Sound \$ specified is a filename. A good programming practice is to include the full path in the filename specified. |
| Result | If the sound is played, this integer variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

Before PiXCL can play a sound file, you must first install a supported sound board and its associated waveform audio device driver. In addition, the sound file must fit in available physical memory and be playable by the installed waveform audio device driver.

If a sound card is not fitted or enabled, this command has no effect.

The [WAVPlaySound](#) function also plays waveform sounds referred to by a keyname in the Sounds section of the registry. (You can see this section of the registry by starting the Registry Editor [REGEDT32.EXE in the Windows NT and 95 system directory] and selecting the tree HKEY_CURRENT_USER on Local Machine.) For example, the Sounds section of the registry might look like this:

```
SystemAsterisk:REG_SZ:chord.wav,Asterisk
SystemDefault:REG_SZ:ding.wav,Default Beep
SystemExclamation:REG_SZ:chord.wav,Exclamation
SystemExit:REG_SZ:chimes.wav,Windows Logoff
SystemHand:REG_SZ:chord.wav,Critical Stop
SystemQuestion:REG_SZ:chord.wav,Question
SystemStart:REG_SZ:tada.wav,Windows Logon
```

To play a sound identified by a registry entry, call [WAVPlaySound](#) with [Sound](#)\$, set to a string that contains the name of the entry that identifies the sound. For example, to play the sound associated with the SystemStart entry and wait for the sound to finish

before returning, use the following statement:

```
WAVPlaySound("SystemStart", SYNC, DEFAULT, Result)
```

The directories searched for sound files are (in order)

1. The current directory.
2. The Windows directory.
3. The Windows system directory.
4. The directories listed in the PATH environment variable.

Example:

This example plays the [tada.wav](#) sound file and doesn't return until the sound is finished. It then plays the sound associated with the SystemExclamation setting in the registry, which is by default [ding.wav](#).

```
WAVPlaySound("tada.wav", SYNC, FILENAME, Result)  
WAVPlaySound("SystemExclamation", ASYNC, ALIAS, Result)
```

Related Commands:

[MessageBeep](#) [WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPitch](#) [WAVGetPlayRate](#) [WAVSetPitch](#) [WAVSetPlayRate](#)

WAVSetPitch

Sound cards such as the industry standard SoundBlaster (™ Creative Labs) may support playback pitch (i.e. the sound playback frequency) changes. This command set the current pitch setting as a percentage. The default value is 100. Values can be less (e.g. 80) or more (e.g. 130).

Syntax: [WAVSetPitch\(Device,Pitch\)](#)

Parameters:

[Device](#) The device number of the sound device. For a system with one sound card, this will be 0.

[Pitch](#) The default value is 100, and the WAV file will play as recorded. If the operation fails, the value may be

0 = could not open the play device.

-1 = INVALID DEVICE specified

-2 = NO DRIVER loaded

-3 = NO MEMORY for operation

-4 = function NOT SUPPORTED

Remarks:

Not all sound cards support Pitch control. Use the [WAVGetDevCaps](#) command to list the available controls for your card.

Related Commands:

[WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPitch](#) [WAVGetPlayRate](#) [WAVPlaySound](#) [WAVSetPlayRate](#)

WAVSetPlayRate

Sound cards such as the industry standard SoundBlaster (TM Creative Labs) may support playback rate control. This command sets the current rate setting as a percentage. The default value is 100.

Syntax: [WAVGetPlayRate\(Device,PlayRate\)](#)

Parameters:

[Device](#) The device number of the sound device. For a system with one sound card, this will be 0.

[PlayRate](#) The default value is 100, and the WAV file will play as recorded.

If the operation fails, the value may be

0 = could not open the play device.

-1 = INVALID DEVICE specified

-2 = NO DRIVER loaded

-3 = NO MEMORY for operation

-4 = function NOT SUPPORTED

Remarks:

Not all sound cards support playback rate control. Use the [WAVGetDevCaps](#) command to list the available controls for your card.

Related Commands:

[WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPitch](#) [WAVGetPlayRate](#) [WAVPlaySound](#) [WAVSetPitch](#)

WAVSetVolume

Sound cards such as the industry standard SoundBlaster (TM Creative Labs) may support volume control. This command sets the current volume setting as a percentage. The default value is 50, minimum value is 0, and maximum value is 100.

Syntax: [WAVSetVolume\(Device,LeftVol, RightVol,Result\)](#)

Parameters:

[Device](#) The device number of the sound device. For a system with one sound card, this will be 0.

[LeftVol, RightVol](#) The default value is 50, and the WAV file will play as recorded.

[Result](#) 1 if the operation was successful. If the operation fails, the value may be
0 = could not open the play device.
-1 = INVALID DEVICE specified
-2 = NO DRIVER loaded
-3 = NO MEMORY for operation
-4 = function NOT SUPPORTED

Remarks:

Not all sound cards support volume control. Use the [WAVGetDevCaps](#) command to list the available controls for your card.

Related Commands:

[WAVGetDevCaps](#) [WAVGetNumDevs](#) [WAVGetPitch](#) [WAVPlaySound](#) [WAVSetPitch](#) [WAVSetPlayRate](#)

While Loops

PiXCL supports the structured WHILE loop, as follows.

Syntax: `While` num_equality | string_equality
... commands
 If <condition> **Then Break** {optional}
... commands
`EndWhile`

Example:

While supports numeric or string equality tests. You must ensure that the testing variable is initialized before the loop is started, or a syntax error occurs. For example, a While loop that tests a numeric variable ...

```
Count = 0
While Count <= 5
  ...commands
  Count++
EndWhile
```

and a `While` loop that tests a string variable ...

```
Count$ = "A"
While Count = "A"
  ...commands
  If <condition> Then Count$ = "B"
  ... commands
EndWhile
```

See the sample program **ForWhile.pxl** for a detailed example.

Remarks:

PiXCL commands within a `While` loop do not need to be indented, but we recommend that that you do for clarity.

Inserting a `Break` command within a `For-Next` loop will break out of the loop. If a `Break` command is located outside of a `For-Next` or `While` loop, it is ignored.

You can have `GoSub` commands within `While-EndWhile` structures, so long as the subroutine does in fact return to the structure. A `GoTo` statement within the `While-EndWhile` is acceptable, but a `GoTo` that jumps outside the structure is not only poor programming practice, but may eventually cause a syntax error in code that previously runs correctly. **This is not a bug: its been intentionally left in the parser code so a Syntax Error will be flagged.** The Syntax Error occurs because PiXCL keeps a record of the embedded pointers to `EndWhile` keywords, and resets them as the code is executed. A jump outside the structure leaves a valid `EndWhile` pointer, and eventually all the entries are used up.

The same situation applies to using a `GoTo` to jump out of a `If-Else-Endif` loop.

Related commands:

[For-Next](#)

WinAdjustRect

One of the first commands in a PiXCL script is [WinLocate](#) that sets the position of the window. Because your PC may be set to small or large fonts, the size of the titlebar and menu bar (if present), plus the border style defines the size of the client area.

It is often handy to be able to specify the desired client area position and dimensions, without having to be concerned with the actual window size. The [WinAdjustRect](#) command provides this facility. You specify the screen coordinates of the desired client area, and the [WinAdjustRect](#) returns the required coordinates for the [WinLocate](#) command.

Syntax: [WinAdjustRect\(cx1,cy1,cx2,cy2,NOMENU | MENU, wx1,wy1,wx2,wy2\)](#)

Parameters:

[cx1,cy1,cx2,cy2](#) The desired client area in **screen** coordinates.

[NOMENU | MENU](#) Defines if the application has a menubar using the [SetMenu](#) command.

[wx1,wy1,wx2,wy2](#) The returned window area in **screen** coordinates.

Remarks:

The returned window area coordinates can be negative. For example, a client area of 0,0,300,200 will produce a window with negative top left coordinates. A following [WinLocate](#) command will set the window to these negative coordinates.

Example:

`Initialize:`

```
Title$ = "PiXCL 4.2 Image Processing"
UseCaption(Title$)
WinAdjustRect(50,100,880,635,MENU,wx1,wy1,wx2,wy2)
WinLocate(Title$,wx1,wy1,wx2,wy2,Res)
. . .
```

Related Commands:

[WinLocate](#) , [WinGetClientRect](#) , [WinGetLocation](#)

WinClose

Closes an active application.

Syntax: `WinClose(Windowname$,Result)`

Parameters:

Windowname\$ The name of the application you want to check. Be sure to specify the full window name exactly as it appears in the Task List, paying careful attention to spacing (case doesn't matter).

Result An integer variable indicating whether `Windowname$` has been closed. If it was closed, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0.

Remark:

PiXCL is quite picky about `Windowname$`. If you don't enter the window's name exactly as it appears in the title bar, paying careful attention to case and spacing, the function will fail and return a `Result` of zero.

Example:

The following program tries to close a copy of Notepad (one that has NETWORKS.TXT open) and displays a message to that effect.

```
App$ ="Notepad - NETWORKS.TXT"
WinClose(App$,Result)
If Result = 1
    TextOut$ = App$ + " was closed."
Else
    TextOut$ = App$ + " was not closed."
Endif
DrawText(10,10,TextOut$)
WaitInput()
```

Related Commands:

[EnumWindows](#) [WinGetActive](#) [WinSetActive](#) [WinGetLocation](#) [WinLocate](#) [WinShow](#) [WinClose](#)

WinDisableInput

PIXCL 5 command. At times you may need to keep your application running, but disable the keyboard and mouse inputs. The [WinDisableInput](#) command provides the way to do this. This command works on the current PIXCL application only.

Syntax: [WinDisableInput](#)(*Time*, *Result*)

Parameters:

Time 0 = the default condition with all input enabled.
 1 = keyboard and mouse input is disabled until re-enabled.
 >1 = the time in mS that inputs will be disabled.

Result A non-zero value if the operation was successful, otherwise 0.

Remarks:

This command is best used to disable keyboard and mouse while a critical process that you don't want interrupted is running. E.g.

```
WinDisableInput(1,Res)
. . . some processing code
WinDisableInput(0,Res)
```

Related Commands:

[WinShow](#) [WaitInput](#)

WinExist

Determines whether an application is active.

Syntax: [WinExist\(Windowname\\$,Result\)](#)

Parameters:

[Windowname\\$](#) The name of the application you want to check. Be sure to specify the full window name exactly as it appears in the Task List, paying careful attention to spacing (case doesn't matter).

[Result](#) An integer variable indicating whether [Windowname\\$](#) is active. If it is active, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0.

Remark:

PIXCL is quite picky about [Windowname\\$](#). If you don't enter the window's name exactly as it appears in the title bar, paying careful attention to case and spacing, the function will fail and return a Result of zero.

Example:

The following program tests for whether a copy of Notepad (one that has NETWORKS.TXT open) is active and displays a message to that effect.

```
App$="Notepad - NETWORKS.TXT"
WinExist(App$,Result)
If Result = 1 Then TextOut$ = App$ + " is active" | Goto Draw
TextOut$ = App$ + " is not active"
Draw:
DrawText(10,10,TextOut$)
WaitInput()
```

Related Commands:

[WinClose](#) [WinGetActive](#) [WinLocate](#) [WinSetActive](#) [WinShow](#)

WinGetActive

Returns the name of the active application window.

Syntax: [WinGetActive\(Windowname\\$\)](#)

Parameter:

[Windowname\\$](#) A string variable that will contain the name of the active application window as it appears in the application's title bar.

Remark:

The name that PiXCL returns is the one that appears in the active application title bar. In the multi-tasking environment of Windows 95 and NT, this may not always get you the window that you expect. This is because other applications may get their slice of cpu time just after the PiXCL application issues the WinGetActive command to Windows. If this is a problem in your PiXCL application, try to write your script to verify that the partial name of the located window. This is usually easy to do. For example, the name of the application, or the file that is being handled can often be used as the check.

Example:

The following program gets the name of the active application and minimizes its window to an icon.

```
WinGetActive(Windowname$)
WinShow(Windowname$,MINIMIZE,Result)
```

Related Commands:

[WinClose](#) [WinSetActive](#) [WinLocate](#) [WinShow](#)

WinGetClientRect

The [WinGetClientRect](#) function retrieves the coordinates of a window's client area. The client coordinates specify the upper-left and lower-right corners of the client area. Because client coordinates are relative to the upper-left corner of a window's client area, the coordinates of the upper-left corner are (0,0).

Syntax [WinGetClientRect](#)(*Window\$,x1,y1,x2,y2*)

Parameters

[Window\\$](#) The name of the target window. This can be acquired from several commands. If [Window\\$](#) is NULL (i.e. "") the current PiXCL application window name is used.

[x1,y1,x2,y2](#) The returned client area coordinates of the selected window. [x1, y1](#) always return zero

Remarks

If the Window does not exist, [x2,y2](#) will also return zero. This command is useful in determining if the system is using small or large fonts, as this affects the number of absolute vertical screen pixels used for the title and menu bars.

The returned coordinates can be passed directly to any of the [SetMouse](#) commands, when the whole client area is required to be mouse active.

Example:

```
WinGetClientRect (Win$, cx1, cy1, cx2, cy2)
SetMouse (cx1, cy1, cx2, cy2, Process, X, Y)
Goto Wait_for_Input
```

Process:

```
{ handle the mouse click...}
SetMouse()
Goto Wait_for_Input
```

Related Commands:

[EnumWindows](#) [WinShow](#) [WinGetLocation](#) [WinLocate](#) [WinAdjustRect](#)

WinGetLocation

Gets the co-ordinate location of a parent window and returns the upper-left corner of window using (x1,y1) and the lower-right corner using (x2,y2).

Syntax: `WinGetLocation(Windowname$,x1,y1,x2,y2,Result)`

Parameters:

| | |
|---------------------------|---|
| <code>Windowname\$</code> | The name of the application you want to reposition. Be sure to use the full window name exactly as it appears in the Task List, paying careful attention to spacing (case doesn't matter). |
| <code>x1,y1</code> | The upper-left corner of the specified window. |
| <code>x2,y2</code> | The lower-right corner of the specified window. |
| <code>Result</code> | An integer variable indicating whether or not <code>Windowname\$</code> was successfully located. If the co-ordinates can be located, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

By default PiXCL uses metric coordinates. If you want to position windows with a higher degree of precision, use the `UseCoordinates` command to change the unit of measure to pixels.

The returned co-ordinates can be positive or negative, since windows can be moved to a very large co-ordinate space. The top left corner of your screen is co-ordinate (0,0).

Example:

The following program changes the coordinate system to pixels, launches Notepad, and then repositions it elsewhere on the screen, set to TOPMOST display.

```
UseCoordinates (PIXEL)
Run ("Notepad")
WinGetLocation ("Notepad - (Untitled) ", X1, Y1, X2, Y2, Result)
X1 = X1 + 200      Y1 = Y1 + 200
X2 = X2 + 200    Y2 = Y2 + 200
WinLocate ("Notepad - (Untitled) ", X1, Y1, X2, Y2, Result)
WinShow ("Notepad - (Untitled) ", TOPMOST, Result)
```

Related Commands:

[EnumWindows](#) [WinClose](#) [WinGetActive](#) [WinSetActive](#) [WinShow](#) [WinAdjustRect](#)

WinHelp

Run the WinHelp utility to access any Help file, or Helpfile topics by topic number or name or keyword.

Syntax: `WinHelp(HelpFile$,COMMAND_TOKEN,KeyWord$)`

Parameters:

| | |
|----------------------------|--|
| <code>Windowname\$</code> | The name of the help file you want to view. This can be in the current directory, or any directory in the PATH. It is best to specify the exact path in the help filename string. |
| <code>COMMAND_TOKEN</code> | The type of command to be issued to WinHlp32. This can any one of COMMAND, CONTENTS, CONTEXT, HELPONHELP, FINDER, KEY, PARTIALKEY or QUIT. |
| <code>KeyWord\$</code> | This argument varies depending on the COMMAND_TOKEN, as follows: |
| <code>COMMAND</code> | <code>KeyWord\$</code> contains a Windows Help macro or set of macros in one string. For example, <code>KeyWord\$</code> can be set to "About ()" and will display the Help file AboutBox information. See more information on Help Macros . |
| <code>CONTENTS</code> | Display the selected Help file Contents topic. All Help files are supposed to have a Contents topic, even if not labeled as such. <code>KeyWord\$</code> should be set to NULL (i.e. "") for this command. |
| <code>CONTEXT</code> | <code>KeyWord\$</code> is an integer number string that references a particular help topic. It is unlikely that you will have this information unless you are building the help files yourself. This argument can be set to "0" when not required or known. |
| <code>HELPONHELP</code> | Displays the standard Window that provides instructions on how to use Helpfiles. <code>KeyWord\$</code> should be set to NULL (i.e. "") for this command. |
| <code>FINDER</code> | Displays the standard Index and Find property sheets. <code>KeyWord\$</code> should be set to NULL (i.e. "") for this command. |
| <code>KEY</code> | <code>KeyWord\$</code> is the keyword for which WinHlp32 searches. It displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, it displays the Index with the topics listed in the Topics Found list box. To display the Index without passing a keyword, you should set <code>KeyWord\$</code> to a NULL string. Multiple keywords must be separated by semicolons within the string. |
| <code>PARTIALKEY</code> | <code>KeyWord\$</code> is the keyword for which WinHlp32 searches. It displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, it displays the Topics Found dialog box. To display the Index without passing a keyword, you should use a pointer to a NULL string. Multiple keywords must be separated by semicolons. |
| <code>QUIT</code> | Closes the specified Help window if it exists. <code>KeyWord\$</code> should be set to NULL (i.e. "") for this command. |

Starting a Help file at a specific topic using the Run("WinHlp32 ...") method.

WINHLP32 also accepts arguments that allow you to specify a Help file topic to be displayed, either by an integer ID number or symbol, which you won't know unless you are the author of the Help file, and are using a Help authoring tool such as BlueSky ROBOhelp (TM), or by specifying the EXACT topic title string as it appears in the Help file. The arguments are

winhlp32.exe **[-H] [-G[n]] [-W window_name] [-K keyword]**
[-N integer] [-I topic_name] [-P popup-id]
HLP_filename]

| Parameter | Description |
|-----------------------|--|
| -G[n] | Creates a configuration (.gid) file and quits. If a number is specified, it determines which extensible tab to display by default the first time the Help file is opened. A value of 1 would be the first tab beyond the Find tab. |
| -H | Displays the Winhlp32.hlp Help file. |
| -I topic_name | Displays the Help topic with the specified the topic ID. |
| -K keyword | Displays the topic identified by the specified keyword. |
| -N integer | Displays the topic specified by the context number (defined in the [MAP] section of the project file). |
| -P popup_id | Displays the specified pop-up topic. You must use the -P switch in combination with the -I or -N switch, as shown in the following examples: WINHELP -P -I EXEC_WINHELP HCW.HLP WINHELP -P -N 311 MYFILE.HLP |
| -W window_name | Displays the topic in the specified window definition. |
| HLP_filename | Specifies the Help file to display. If a name is not specified, the File Open dialog box appears. |

Comments

The **-G** switch should be used by Setup programs when they install a newer version of a Help file or a contents file. This switch causes the .gid file to be rebuilt. WinHlp32 will automatically build the .gid file the first time a Help file is opened — unless the winhlp32 -G command has already been run on the file.

If the **topic_name** or **keyword** is two or more words, the spaces and any other punctuation characters must be replaced by the underscore character (“_”). Multiple keywords should be separated by a semi-colon.

Here is some PiXCL code that starts a Help file at a specific topic. Observe that the existence of the Help file is first verified.

```
Start:
    HelpFile$ = SourceDir$ + "\newapp.hlp"
    FileExist (HelpFile$,Res)
    If Res = 1 Then Goto See_Help
        MessageBox (OK,1,EXCLAMATION,
            HelpFile$,"WARNING: Unable to locate Help file",Res)
        Goto Wait_For_Input
    {endif}
See_Help:
    CommandLine$ = "WinHlp32 -n 125 " + HelpFile$
    {or use this alternative ...
    CommandLine$ = "WinHlp32 -i Help_topic_Two " + HelpFile$
    ...end of comment}
    Run (CommandLine$)
    Goto Wait_for_Input
```

Notes:

1. While .HLP is one of the default Windows file associations, and the Program Manager (and similar) Run commands will automatically assume that WINHLP32.EXE should be invoked because of in-built file associations, it is not the case with the PiXCL Run(...) command. If you issue a command

```
Run("newapp.hlp")
```

you will get a message box informing you that the "...file cannot be accessed at this time."

Related Command

[WinHTMLHelp](#)

WinHTMLHelp

Run the HTMLHelp utility (if installed) to access any compiled HTML Help file, or Helpfile topics by topic name or keyword. This command in PiXCL 4.20 and later requires HTML Help controls and DLLs. If you have Internet Explorer 4.01 or later installed these controls and DLLs are already present on your PC. There are also found in the Microsoft free download **HHUPD.EXE**, which is available from <http://www.microsoft.com> and from the PiXCL Registered Users' Area. Download these if you don't want to install Internet Explorer 4.01

HTML Help is the new "standard" being promoted by Microsoft, and is at least as capable as the traditional [WinHelp](#) method, which will continue to be supported. Microsoft also has a free HTML help utility called HTML Help Workshop, available at <http://www.microsoft.com/workshop/author/htmlhelp>. Please consult this site or a variety of publications for full details on HTML Help authoring.

Syntax: WinHTMLHelp(*HtmlHelp\$*, MODE_TOKEN, COMMAND_TOKEN, *KeyWord\$*,x1,y1,x2,y2)

Parameters:

HtmlHelp\$

The name of the help file you want to view. This can be in the current directory, or any directory in the PATH. It is best to specify the exact path in the help filename string. The extension is **.chm**.

MODE_TOKEN

The HTMLHelp view mode. This can any one of

SIBLING

Display help in a window that is initially on top of the PiXCL application, but any other window can cover it.

OWNED

Display help in a window that is always on top of the PiXCL application, but not any other application window.

CHILD

Display help in a window that is a child of the PiXCL application.

COMMAND_TOKEN

The type of command to be issued to the HTMLHelp viewer. This can any one of **CONTENTS**, **CONTEXT**, **POPUP**, **KEY** or **QUIT**.

KeyWord\$

This argument varies depending on the **COMMAND_TOKEN**, as follows:

CONTENTS

Display the selected Help file Contents topic. All Help files are supposed to have a Contents topic, even if not labeled as such. *KeyWord\$* should be set to NULL (i.e. "") for this command. Arguments *x1*, *y1*, *x2*, *y2* should be set to zero, and are ignored.

CONTEXT

KeyWord\$ is an integer number string that references a particular help topic. It is unlikely that you will have this information unless you are building the help files yourself, and creating context IDs. This argument can be set to a **null** string when not required or known. *x1* is used for the context ID. *y1*, *x2,y2* should be set to zero, and are ignored.

POPUP

Displays a popup window with the text string defined in *HtmlHelp\$* with the desired font in *KeyWord\$*, in the form "**facename[,point size[,charset[,color[, BOLD ITALIC UNDERLINE]]]]**".

Note there are no commas between BOLD ITALIC UNDERLINE.

Charset is either left blank or set to 0. Values of 1 and 2 may produce alternative character sets for the selected font.

Color takes the form **#00bbggrr**. This is the same format for colours as in the HTML keyword ****. Hence red is defined as **#000000ff**.

If the parameter is not used e.g. **charset**, it is left blank, but the comma is still required. This command is very handy because you can create HTML string variables within your PiXCL application, without the need for external files. The windows top-center screen coordinates are *x1,y1*. Arguments *x2,y2* should be set to zero, and are ignored. If *KeyWord\$* is NULL, the current font and popup text color are set by a **UseFont** command is used. If no font is selected, the default is "Arial,,,". **Note the multiple commas.**

KEY

KeyWord\$ is the keyword for which HTML Help searches. It displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, it displays the Index with the topics listed in the Topics Found list box. To display the Index without passing a keyword, you should set *KeyWord\$* to a NULL string. Multiple keywords must be separated by

semicolons within the string. Arguments *x1,y1*, *x2,y2* should be set to zero, and are ignored.

QUIT

Closes the specified Help window if it exists. *KeyWord\$* should be set to NULL (i.e. "") for this command. Arguments *x1,y1*, *x2,y2* should be set to zero, and are ignored.

x1, y1, x2, y2

Used for general purpose position and values, depending on the command.

Examples.

Here is some PiXCL code that starts a Help file at a specific topic. Observe that the existence of the Help file is first verified.

Start:

```
HTMLHelpFile$ = SourceDir$ + "\newapp.chm"
FileExist(HTMLHelpFile$,Res)
If Res = 1 Then Goto See_Help
    MessageBox(OK,1,EXCLAMATION,
        HelpFile$,"WARNING: Unable to locate HTML Help file",Res)
    Goto Wait_For_Input
{endif}
```

See_Help:

```
WinHTMLHelp(HTMLHelpFile$,SIBLING,CONTENTS,"",0,0,0,0)
WinHTMLHelp(HTMLHelpFile$,SIBLING,POPUP,"This is a popup",100,120,0,0)
Goto Wait_for_Input
```

CloseHelp:

```
WinHTMLHelp(HTMLHelpFile$,SIBLING,QUIT,"",0,0,0,0)
Goto Wait_for_Input
```

See also the sample application source, [htmlhelp.pxl](#).

Remarks:

1. If you don't close the HTML Help file, PiXCL will unload the HTML Viewer library.
2. Popup windows will be cleared on the next left or right mouse click.
3. If you invoke WinHTMLhelp from a toolbar button, it is VERY advisable to include a WaitInput(1) as the first command after the button handler label, so that Windows can catch up.

Related Commands:

[HTMLControl](#)

WinLocate

Positions a window to a new location. You specify the upper-left corner of the new location using (x1,y1) and the lower-right corner using (x2,y2).

Syntax: `WinLocate(Windowname$,x1,y1,x2,y2,Result)`

Parameters:

| | |
|---------------------------|--|
| <code>Windowname\$</code> | The name of the application you want to reposition. Be sure to use the full window name exactly as it appears in the Task List, paying careful attention to spacing (case doesn't matter). |
| <code>x1,y1</code> | The upper-left corner of the window. |
| <code>x2,y2</code> | The lower-right corner of the window. |
| <code>Result</code> | An integer variable indicating whether or not <code>Windowname\$</code> was successfully repositioned. If it was repositioned, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remarks:

By default PiXCL uses metric coordinates. If you want to position windows with a higher degree of precision, use the [UseCoordinates](#) command to change the unit of measure to pixels.

If a window is maximized or minimized, be sure to use the [WinShow](#) command to restore it to its normal size before moving it with [WinLocate](#).

Example:

The following program changes the coordinate system to pixels, launches Notepad, and then repositions it to occupy a small square at the upper-left corner of the screen.

```
UseCoordinates (PIXEL)
Run ("Notepad")
WinLocate("Notepad - (Untitled)",0,0,200,200,Result)
```

Related Commands:

[WinClose](#) [WinGetActive](#) [WinSetActive](#) [WinShow](#) [WinAdjustRect](#)

WinResizeAt

Some applications require idle loop processing that needs to know when the main PiXCL application window has been resized. The [WinResizeAt](#) command provides a jump-to label so processing (usually some sort of client area redraw) can be done as needed, without having an inefficient polling loop. Once set, the jump-to label can only be changed, not disabled. You can set the jump-to label to a dummy label or the main idle loop label if required.

Syntax: [WinResizeAt\(Label\)](#)

Parameter:

[Label](#) The jump-to label. If [Label](#) does not exist in the script, a syntax error is generated.

Related Commands:

[BMWSysCmdEndAt](#) [SysCmdEndAt](#)

WinSetActive

Activates an application.

Syntax: `WinSetActive(Windowname$,Result)`

Parameters:

Windowname\$ The name of the application you want to make active. Be sure to use the full window name exactly as it appears in the Task List, paying careful attention to spacing (case doesn't matter).

Result An integer variable indicating whether or not `Windowname$` was successfully activated. If it was activated, this variable is assigned a value of 1. Otherwise, this variable is assigned a value of 0.

Remarks:

When using this command, be aware that window names can change. Most applications append the name of the working file to the end of the application name in the window title bar. For example, Notepad might have window names like

Notepad - (Untitled)
Notepad - README.TXT

Example:

The following program gets the name of the PiXCL window and then launches three applications in succession: Notepad, Write, and Cardfile. After launching each application, it stores its name in a string variable—for example, Win1\$ is assigned "Notepad - (Untitled)". Next, it reactivates the PiXCL window (bringing it to the front of the stack), puts up a list box containing the names of the three applications, and prompts you to choose one to activate. After you select a name from the list followed by OK, the program activates the chosen application.

```
WinSetActive (PiXCL$)
Run ("Notepad")
WinSetActive (Win1$)
Run ("Write")
WinSetActive (Win2$)
Run ("Cardfile")
WinSetActive (Win3$)
Caption$ = "Choose an application to activate"
List$ = Win1$ + ";"
List$ = List$ + Win2$
List$ = List$ + ";"
List$ = List$ + Win3$
WinSetActive (PiXCL$, Ignore)
ListBox (Caption$, List$, ";", Window$)
If Window$ = "" Then End
WinSetActive (Window$, Ignore)
WaitInput ()
```

Related Commands:

[WinClose](#) [WinGetActive](#) [WinLocate](#) [WinShow](#)

WinShow

Hides, unhides, minimizes, maximizes, or restores an application window.

Syntax:

```
WinShow(Windowname$,  
        HIDE / UNHIDE/ MINIMIZE / MAXIMIZE /  
        RESTORE / TOP/ TOPMOST / NOTOPMOST /  
        BOTTOM / SHOWINNOACTIVE / SHOWNOACTIVATE /  
        NOMOVE / NOSIZE / NOTITLE,  
        Result)
```

Parameters:

| | |
|-----------------------------|--|
| <code>Windowname\$</code> | The name of the application whose window you want to affect. |
| <code>HIDE</code> | Hides the window. |
| <code>UNHIDE or SHOW</code> | Unhides the window. |
| <code>MAXIMIZE</code> | Maximizes the window. |
| <code>MINIMIZE</code> | Minimizes the window. |
| <code>RESTORE</code> | Restores the window. |
| <code>TOP</code> | Sets the window to the top of the Z-order list. |
| <code>TOPMOST</code> | Sets the window to always topmost window |
| <code>NOTOPMOST</code> | Resets the TOPMOST flag |
| <code>BOTTOM</code> | Move the window to the bottom of the Z-order list |
| <code>SHOWINNOACTIVE</code> | Minimize the window but don't change the focus. |
| <code>SHOWNOACTIVATE</code> | Display the window in the last current position but don't change the focus. |
| <code>NOMOVE</code> | For PiXCL applications only: removes the title bar and sets the window style so that it cannot be moved except with the WinLocate command. |
| <code>NOSIZE</code> | For PiXCL applications only: sets the border style to thin, preventing the window being sized with the mouse. To size, use the WinLocate command. |
| <code>NOTITLE</code> | For PiXCL applications only: removes the title bar from the displayed window. It is still possible to resize the window and move it with the WinLocate command. |
| <code>Result</code> | An integer variable indicating the outcome of the operation. If the display characteristics of the window were successfully changed, this variable is assigned a value of 1. Otherwise, it is assigned a value of 0. |

Remark:

You can use this command to change the display characteristics of any running application, including the PiXCL window. Using `NOMOVE`, `NOSIZE` and `NOTITLE` has no effect on other application windows: i.e. the commands affect only the current running PiXCL application that includes the command.

Example:

This example gets the name of the PiXCL window and then displays a message box asking whether you want to hide the window. If you select Yes, the window immediately disappears from view and reappears after two seconds.

```
WinGetActive(Win$)
```

```
MessageBox (YESNO,1,QUESTION,
            "Hide window for 2 seconds?","Hide",Button)
If Button = 2 Then End
WinShow (Win$,HIDE,Result)
WaitInput (2000)
WinShow (Win$,UNHIDE,Result)
DrawText (10,10,"We're back")
WaitInput ()
```

The HIDE token should be used with care. If you HIDE a window, and don't UNHIDE it later, it will be completely hidden from access, even though it is still running. The Windows task list will not show the application, and you will not be able to stop it unless you log out of the system. You can recover from this by writing another PiXCL program that uses the [EnumWindows\(...\)](#) command, then [WinShow\(Win\\$, UNHIDE,Result\)](#).

Related Commands:

[EnumWindows](#) [WinClose](#) [WinGetActive](#) [WinSetActive](#) [WinGetLocation](#) [WinLocate](#) [WinShowLayered](#)

WinShowLayered

PIXCL 5 command. For Windows 2000 systems only, you can set either a transparency color or alpha blend level to display what is called a **layered window**. This command has no effect under Windows 9x/ME/NT4, as the layering functions are not present in Windows itself.

Syntax: *WinShowLayered(WindowName\$,R,G,B, AlphaLevel, modeTOKEN,Result)*

Parameters:

| | |
|---------------------|--|
| <i>WindowName\$</i> | The target window. |
| <i>R,G,B</i> | The RGB transparency colour, if COLORKEY mode is selected. |
| <i>AlphaLevel</i> | A value between 0 = completely transparent, to 255 = completely opaque, if ALPHA mode is selected. |
| COLORKEY | Use the RGB value to set the transparency colour. |
| ALPHA | Display the window at the alpha value. |
| UNLAYER | Turn off the transparency or alpha blending. |
| <i>Result</i> | 1 if the operation was successful, otherwise 0. |

Related Command:

[WinShow](#)

WinTitle

Sets the text that appears in the title bar of a main window.

Syntax: `WinTitle(Windowname$, Title$)`

Parameters:

`Windowname$` The name of the application whose title bar you want to change.

`Title$` The text you want to appear in the title bar.

Remark:

If you want to use this command to change the caption of the PiXCL window, be sure to provide the current caption in its entirety for `WindowName$`--for example, "PiXCL44 - test.pxI".

Example:

This example starts Notepad and then changes its caption to "George's Editor":

```
Run("Notepad")
WinTitle("Notepad - (Untitled)", "George's Editor")
```

Related Command:

[UseCaption](#)

WinVersion

Returns the current Windows version number.

Syntax: *WinVersion(Major,Minor,Build,ServicePack\$)*

Parameters:

| | |
|----------------------|---|
| <i>Major</i> | The version number: 4 for Windows 95, 98, ME, NT4 and 5 for 2000. |
| <i>Minor</i> | The decimal portion of the version number: 00 for Windows 95, NT4 and 2000, 10 for Windows 98, and 90 for Windows ME. |
| <i>Build</i> | This is the build number of the specific version. Build numbers are specially relevant during the Microsoft pre-release programs. They can be useful in locating issues between different Win32 releases, for example between Windows NT, Windows 95 and Windows 98. In Windows 95, this is reported as 950, in Windows 98, as 1998. OEM versions of Windows 98 report other numbers: Dell systems for example will often report 2222AA which indicates Windows 98 2nd edition. In NT 4 Workstation, this is usually 1381 for the English language version. The point is that the build number is quite variable, and is provided for information purposes only, or where a system administration task requires it. |
| <i>ServicePack\$</i> | In Windows NT, this string reports the installed service pack, e.g. "Service Pack 3". In Windows 95 and 98, this string will generally be null, but may report arbitrary information about the software version. |

Example:

If you're writing an PiXCL program that you expect will be used on Windows 95, 98, NT4 and Windows 2000 systems, you can tailor your font selection accordingly. For example, this program determines the Windows version number and then uses Arial for Windows NT 4, Roman for Windows 95, and Garamond for Windows 98. It then displays some sample text using the chosen font.

```
WinVersion(Major,Decimal,Build,Pack$)
If Build = 0950
    UseFont("Roman",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
    SampleText$="This is Windows 95's Roman"
Endif
If Build = 1998
    UseFont("Garamond",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
    SampleText$="This is Windows 98's Garamond"
Endif
If Build = 1381
    UseFont("Arial",0,0,NOBOLD,NOITALIC,NOUNDERLINE,0,0,0)
    SampleText$="This is Windows NT4's Arial"
Endif
DrawText(10,10,SampleText$)
```

WriteBitmapID

PIXCL 5 command. Any bitmap loaded into the PiXCL image list can have an identifier string written into the bitmap data at an arbitrary location. Because strings are ascii characters in the range 0-127 and 128-255 for other characters, an identifier string will often be in effect invisible. The general term for this hiding of data within other data is steganography.

Syntax: [WriteBitmapID\(Imagename\\$,Pixel,Line,Idstring\\$,Result\)](#)

Parameters:

| | |
|-----------------------------|---|
| Imagename\$ | The image loaded into the PiXCL image list. If the image is not loaded, the function fails. |
| Pixel, Line | The start coordinate for the ID string. |
| Idstring\$ | The string to write. This must fit into the current line. |
| Result | 1 if the operation succeeded, otherwise 0. |

Related Command:

[ReadBitmapID](#) [SaveBitmap](#)

WriteBitmapRect

Write a sub-area of a BMP bitmap on the disk, using the current image in the PiXCL image list. The current image must have the same bits per pixel as the target bitmap.

Syntax: `WriteBitmapRect(ImageFile$,x1,y1,x2,y2,Result)`

Parameters:

ImageFile\$ The target image on the disk. Your PiXCL code must ensure that you are accessing the current image in the list.

x1,y1,x2,y2 The region in the target bitmap on disk that is to be replaced.

Result 1 if the operation is successful, otherwise 0.

Example:

This code fragment reads an image rectangle into memory, then writes it out to the target bitmap, after converting the subarea into 24 bits per pixel.

ProcessSubArea:

```
ReadBitmapRect (Image9$, 0, 0, 100, 100, Res)
DrawBitmap(100,100, Image9$)
DrawStatusWinText (0, Image9$)
ConvertColorSpace (RGB24, NONE, Res)
WriteBitmapRect (Image5$, 0, 0, 100, 100, Res)
If Res = 0 Then DrawStatusWinText(0, "Write Rectangle failed.")
FreeBitmapAll
DrawBitmap(60, 60, Image5$)
Goto Wait_for_Input
```

Related Commands:

[ReadBitmapRect](#)

WriteCommPort

Asynchronous data streams typically include variable length data, followed by a terminating character or string. Most useful serial devices like digitizing tablets let you configure the terminating string. In the current version of PiXCL, this string MUST be set to a carriage return and linefeed pair (CR-LF).

PiXCL maintains a write buffer of about 4 KB. When the last character in the buffer is written to the port, a comms event is generated to which your PiXCL program can respond, in much the same way that your program responds to mouse or keyboard events in the WaitInput() idle loop.

Hence, to receive a write comms event, you would use a command [WaitCommsEvent\(W,<label>,Timeout\)](#)

Syntax: [WriteCommPort\(COMx,Data\\$\)](#)

Parameters:

[COMx](#) Port, where x = 1 - 4

[Data\\$](#) The data written to the port buffer.

Related Commands:

[ClearCommPort](#) [EscCommFunction](#) [GetCommPort](#) [ReadCommPort](#) [SetCommPort](#) [WaitCommEvent](#)

ZoomBitmapWindow

You can set the positive or negative zoom factor of the image in a bitmap window with this command. The minimum zoom factor available is 16:1, which is reported by the [GetBMWZoom](#) command as "[063:1]".

Syntax: [ZoomBitmapWindow](#)(*WindowId*,*ImageFile\$*, *xPos*,*yPos*, *Factor*,
RELATIVE|INCREMENT|ABSOLUTE)

Parameters:

| | |
|---------------------------|--|
| WindowID | The ID number returned by the DrawBitmapWindow command. |
| xPos,yPos | The BITMAP coordinates to zoom on, which may have been returned by the SetBMWMouse or SetBMWRightMouse command. The command will try to put this point in the center of the bitmap window. |
| Factor | The new zoom factor in the range -16 to -2 and +1 to +16, multiplied by 10. Hence a zoom factor of 2 requires a Factor of 20. A value of -1 or 0 is equivalent to +1. |
| RELATIVE | Zoom relative to the current factor. |
| INCREMENT | Zoom from the current factor. |
| ABSOLUTE | Zoom absolute mode. |

Related Commands:

[BMWinTitle](#) [CloseBitmapWindow](#) [DrawBitmapWindow](#) [GetBMWZoom](#) [SetBMWMouse](#)

MCI Command Strings

The following command strings are used with MCI devices.

| | |
|-----------------------------------|------------------------------------|
| <u>break</u> | <u>realize</u> |
| <u>capability</u> | <u>record</u> |
| <u>capture</u> | <u>reserve</u> |
| <u>close</u> | <u>restore</u> |
| <u>configure</u> | <u>resume</u> |
| <u>copy</u> | <u>save</u> |
| <u>cue</u> | <u>seek</u> |
| <u>cut</u> | <u>set</u> |
| <u>delete</u> | <u>setaudio</u> |
| <u>escape</u> | <u>settimecode</u> |
| <u>freeze</u> | <u>settuner</u> |
| <u>index</u> | <u>setvideo</u> |
| <u>info</u> | <u>signal</u> |
| <u>list</u> | <u>spin</u> |
| <u>load</u> | <u>status</u> |
| <u>mark</u> | <u>step</u> |
| <u>monitor</u> | <u>stop</u> |
| <u>open</u> | <u>sysinfo</u> |
| <u>paste</u> | <u>undo</u> |
| <u>pause</u> | <u>unfreeze</u> |
| <u>play</u> | <u>update</u> |
| <u>put</u> | <u>where</u> |
| <u>quality</u> | <u>window</u> |

break

The **break** command specifies a key to abort a command that was invoked using the "wait" flag. This command is an MCI system command; it is interpreted directly by MCI.

MCI Syntax: "break DeviceID VirtualKey Flags"

Parameters:

| | | | | | |
|----------------------------|---|----------------------------|---|-----|---------------------------------|
| DeviceID | Identifier of an MCI device. This identifier or alias is assigned when the device is opened. | | | | |
| VirtualKey | One of the following flags: <table><tr><td>on <i>virtual key code</i></td><td>Specifies the key that aborts a command that was started using the "wait" flag.</td></tr><tr><td>off</td><td>Disables the current break key.</td></tr></table> | on <i>virtual key code</i> | Specifies the key that aborts a command that was started using the "wait" flag. | off | Disables the current break key. |
| on <i>virtual key code</i> | Specifies the key that aborts a command that was started using the "wait" flag. | | | | |
| off | Disables the current break key. | | | | |
| Flags | Can be "wait", "notify", or both. For digital-video and VCR devices, "test" can also be specified. | | | | |

capability

The **capability** command requests information about a particular capability of a device. All MCI devices recognize this command.

MCI Syntax: "capability DeviceID Request Flags"

Parameters:

DeviceID Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Request Flag that identifies a device capability. The following table lists device types that recognize the **capability** command and the flags used by each type:

| | | |
|---------------------|---------------------|----------------------|
| cdaudio | can eject | device type |
| | can play | has audio |
| | can record | has video |
| | can save | uses files |
| | compound device | |
| digitalvideo | can eject | compound device |
| | can freeze | device type |
| | can lock | has audio |
| | can play | has still |
| | can record | has video |
| | can reverse | maximum play rate |
| | can save | minimum play rate |
| | can stretch | uses files |
| | can stretch input | uses palettes |
| | can test | windows |
| overlay | can eject | compound device |
| | can freeze | device type |
| | can play | has audio |
| | can record | has video |
| | can save | uses files |
| sequencer | can stretch | windows |
| | can eject | device type |
| | can play | has audio |
| | can record | has video |
| | can save | uses files |
| vcr | compound device | |
| | can detect length | clock increment rate |
| | can eject | compound device |
| | can freeze | device type |
| | can monitor sources | has audio |
| | can play | has clock |
| | can preroll | has timecode |
| | can preview | has video |
| | can record | number of marks |
| | can reverse | seek accuracy |
| | can save | uses files |
| can test | | |
| videodisc | can eject | device type |
| | can play | fast play rate |
| | can record | has audio |
| | can reverse | has video |
| | can save | normal play rate |
| | CAV | slow play rate |
| | CLV | uses files |
| | compound device | |
| waveaudio | can eject | has audio |
| | can play | has video |
| | can record | inputs |
| | can save | outputs |
| | compound device | uses files |
| | device type | |

The following table lists the flags that can be specified in the *Request* parameter and their meanings:

can detect length Returns TRUE if the device can detect the length of the media.

| | |
|-----------------------------|---|
| can eject | Returns TRUE if the device can eject the media. |
| can freeze | Returns TRUE if the device can freeze data in the frame buffer. |
| can lock | Returns TRUE if the device can lock data. |
| can monitor sources | Returns TRUE if the device can pass an input (source) to the monitored output, independent of the current input selection. |
| can play | Returns TRUE if the device can play. |
| can preroll | Returns TRUE if the device supports the "preroll" flag with the cue command. |
| can preview | Returns TRUE if the device supports previews. |
| can record | Returns TRUE if the device supports recording. |
| can reverse | Returns TRUE if the device can play in reverse. |
| can save | Returns TRUE if the device can save data. |
| can stretch | Returns TRUE if the device can stretch frames to fill a given display rectangle. |
| can stretch input | Returns TRUE if the device can resize an image in the process of digitizing it into the frame buffer. |
| can test | Returns TRUE if the device recognizes the test keyword. |
| cav | When combined with other items, this flag specifies that the return information applies to CAV format videodiscs. This is the default if no videodisc is inserted. |
| clock increment rate | Returns the number of subdivisions the external clock supports per second. If the external clock is a millisecond clock, the return value is 1000. If the return value is 0, no clock is supported. |
| clv | When combined with other items, this flag specifies that the return information applies to CLV format videodiscs. |
| compound device | Returns TRUE if the device supports an element name (filename). |
| device type | Returns a device type name, which can be one of the following: cdaudio dat digitalvideo other overlay scanner sequencer vcr videodisc waveaudio |
| fast play rate | Returns the fast play rate in frames per second, or zero if the device cannot play fast. |
| has audio | Returns TRUE if the device supports audio playback. |
| has clock | Returns TRUE if the device has a clock. |
| has still | Returns TRUE if the device treats files with a single image more efficiently than motion video files. |
| has timecode | Returns TRUE if the device is capable of supporting timecode, or if it is unknown. |
| has video | Returns TRUE if the device supports video. |
| inputs | Returns the total number of input devices. |
| maximum play rate | Returns the maximum play rate, in frames per second, for the device. |
| minimum play rate | Returns the minimum play rate, in frames per second, for the device. |
| normal play rate | Returns the normal play rate, in frames per second, for the device. |
| number of marks | Returns the maximum number of marks that can be used; zero indicates that marks are unsupported. |
| outputs | Returns the total number of output devices. |
| seek accuracy | Returns the expected accuracy of a search in frames; 0 indicates that the device is frame accurate, 1 indicates that the device expects to be within one frame of the indicated seek position, and so on. |

slow play rate

Returns the slow play rate in frames per second, or zero if the device cannot play slowly.

uses files

Returns TRUE if the data storage used by a compound device is a file.

uses palettes

Returns TRUE if the device uses palettes.

windows

Returns the number of simultaneous display windows the device can support.

Flags

Can be "wait", "notify", or both. For digital-video and VCR devices, "test" can also be specified.

capture

The **capture** command copies the contents of the frame buffer and stores it in the specified file. Digital-video devices recognize this command.

MCI Syntax: "capture DeviceID Capture Flags"

Parameters

DeviceID

Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Capture

One or more of the following flags:

as pathname Specifies the destination path and filename for the captured image. This flag is required.

at rectangle Specifies the rectangular region within the frame buffer that the device crops and saves to disk. If omitted, the cropped region defaults to the rectangle specified or defaulted on a previous put "source" command for this device instance.

Flags

Can be "wait", "notify", "test", or a combination of these

Remarks

This command might fail if the device is currently playing motion video or executing some other resource-intensive operation. If the frame buffer is being updated in real time, the updating momentarily pauses so that a complete image is captured. If the device pauses the updating, there might be a visual or audible effect. If the file format, compression algorithm, and quality levels have not been set, their defaults are used.

close

The **close** command closes the device or file and any associated resources. MCI unloads a device when all instances of the device or all files are closed. All MCI devices recognize this command.

MCI Syntax: "close DeviceID Flags"

Parameters

DeviceID Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Flags Can be "wait", "notify", "test", or a combination of these

Remarks

To close all devices opened by your application, specify the "all" device identifier for the DeviceID parameter.

The following command closes the "mysound" device:

close mysound

configure

The **configure** command displays a dialog box used to configure the device. Digital-video devices recognize this command.

MCI Syntax: "configure DeviceID Flags"

Parameters

DeviceID Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Flags Can be "wait", "notify", "test", or a combination of these

copy

The **copy** command copies data to the clipboard. Digital-video devices recognize this command.

MCI Syntax: "copy DeviceID Item Flags"

Parameters

DeviceID

Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Item

One of the following flags identifying the item to copy:

at rectangle Specifies the portion of each frame that will be copied. If omitted, the default setting is the entire frame.

audio stream stream Specifies the audio stream in the workspace affected by the command. If you use this flag and also want to copy video, you must also use the "**video stream**" flag. (If neither flag is specified, all audio and video streams are copied.)

from position Specifies the start of the range copied. If omitted, the default setting is the current position.

to position Specifies the end of the range copied. The audio and video data copied are exclusive of this position. If omitted, the default setting is the end of the workspace.

video stream stream Specifies the video stream in the workspace affected by the command. If you use this flag and also want to copy audio, you must also use the "**audio stream**" flag. (If neither flag is specified, all audio and video streams are copied.)

Flags

Can be "**wait**", "**notify**", "**test**", or a combination of these

cue

The **cue** command prepares for playing or recording. Digital-video, VCR, and waveform-audio devices recognize this command.

MCI Syntax: "cue DeviceID InOutTo Flags"

Parameters

DeviceID

Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

InOutTo

Flag that prepares a device for playing or recording. The following table lists device types that recognize the cue command and the flags used by each type:

| | | |
|---------------------|----------------------|--------------------|
| digitalvideo | input | output |
| | noshow | to position |
| vcr | from position | preroll |
| | input | reverse |
| | output | to position |
| waveaudio | input | output |

The following table lists the flags that can be specified in the **InOutTo** parameter and their meanings:

| | |
|----------------------|---|
| from position | Indicates where to start. |
| input | Prepares for recording. For digital-video devices, this flag can be omitted if the current presentation source is already the external input. |
| noshow | Prepares for playing a frame without displaying it. When this flag is specified, the display continues to show the image in the frame buffer even though its corresponding frame is not the current position. A subsequent cue command without this flag and without the "to" flag displays the current frame. |
| output | Prepares for playing. If neither "input" nor "output" is specified, the default setting is "output" . |
| preroll | Moves the preroll distance from the in-point. The in-point is the current position, or the position specified by the "from" flag. |
| reverse | Indicates play direction is in reverse (backward). |
| to position | Moves the workspace to the specified position. For VCR devices, this flag indicates where to stop. |

Flags

Can be **"wait"**, **"notify"**, **"test"**, or a combination of these

cut

The **cut** command removes data from the workspace and copies it to the clipboard. Digital-video devices recognize this command.

MCI Syntax: "cut DeviceID Item Flags"

Parameters

DeviceID

Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Item

One of the following flags identifying the item to cut:

at rectangle Specifies the portion of each frame cut. If omitted, it defaults to the entire frame. When this item is specified, frames are not deleted. Instead the area inside the rectangle becomes black.

audio stream stream Specifies the audio stream in the workspace affected by the command. If you use this flag and also want to cut video, you must also use the "**video stream**" flag. (If neither flag is specified, all audio and video streams are cut.)

from position Specifies the start of the range cut. If omitted, it defaults to the current position.

to position Specifies the end of the range cut. The audio and video data cut are exclusive of this position. If omitted it defaults to the end of the workspace.

video stream stream Specifies the video stream in the workspace affected by the command. If you use this flag and also want to cut audio, you must also use the "**audio stream**" flag. (If neither flag is specified, all audio and video streams are cut.)

Flags

Can be "**wait**", "**notify**", "**test**", or a combination of these

delete

The **delete** command deletes a data segment from a file. Digital-video and waveform-audio devices recognize this command.

MCI Syntax: "delete DeviceID Position Flags"

Parameters

DeviceID

Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Position

Flag that identifies a data segment to delete. The following table lists device types that recognize the delete command and the flags used by each type:

| | | |
|---------------------|----------------------------|----------------------------|
| digitalvideo | at rectangle | to position |
| | audio stream stream | video stream stream |
| | from position | |
| waveaudio | from position | to position |

The following table lists the flags that can be specified in the `lpszPosition` parameter and their meanings:

at rectangle Specifies the portion of each frame cut. If omitted, it defaults to the entire frame. When this item is specified, frames are not deleted. Instead the area inside the rectangle becomes black.

audio stream stream Specifies the audio stream in the workspace affected by the command. If you use this flag and also want to cut video, you must also use the "**video stream**" flag. (If neither flag is specified, all audio and video streams are cut.)

from position Specifies the start of the range cut. If omitted, it defaults to the current position.

to position Specifies the end of the range cut. The audio and video data cut are exclusive of this position. If omitted it defaults to the end of the workspace.

video stream stream Specifies the video stream in the workspace affected by the command. If you use this flag and also want to cut audio, you must also use the "**audio stream**" flag. (If neither flag is specified, all audio and video streams are cut.)

Flags

Can be "**wait**", "**notify**", "**test**", or a combination of these.

Remarks

Before issuing any commands that use position values, you should set the desired time format by using the **set** command.

The following command deletes the waveform-audio data from 1 millisecond through 900 milliseconds (assuming the time format is set to milliseconds):

delete mysound from 1 to 900

escape

The **escape** command sends device-specific information to a device. Videodisc devices recognize this command.

MCI Syntax: "escape DeviceID Escape Flags"

Parameters

DeviceID Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

Escape Custom information to send to the device. Consult your device documentation.

Flags Can be "**wait**", "**notify**", or both.

Remarks

The following command sends the escape string "SA" to a videodisc device:

escape videodisc SA

freeze

The **freeze** command freezes video input or video output on a VCR or disables video acquisition to the frame buffer. Digital-video, video-overlay, and VCR devices recognize this command.

MCI Syntax: "freeze DeviceID FreezeFlags Flags"

Parameters

DeviceID

Identifier of an MCI device. This identifier or alias is assigned when the device is opened.

FreezeFlags

Flag that identifies what to freeze. The following table lists device types that recognize the freeze command and the flags used by each type:

| | | |
|---------------------|---------------------|----------------|
| digitalvideo | at rectangle | outside |
| overlay | at rectangle | |
| vcr | field | input |
| | frame | output |

The following table lists the flags that can be specified in the FreezeFlags parameter and their meanings:

| | |
|---------------------|--|
| at rectangle | Specifies the region that will be frozen. For video-overlay devices, this region will have video acquisition disabled. For digital-video devices, the pixels within the rectangle will have their lock mask bit turned on (unless the " outside " flag is specified). The rectangle is relative to the video buffer origin and is specified as X1 Y1 X2 Y2. The coordinates X1 Y1 specify the upper left corner of the rectangle, and the coordinates X2 Y2 specify the width and height. |
| field | Freezes the first field. Field is assumed by default (if neither frame nor field is specified). |
| frame | Freezes the entire frame, displaying both fields. |
| input | Freezes the current frame of the input image, whether it is paused or running. |
| output | Freezes the current frame of the output from the VCR. If the VCR is playing when freeze is issued, the current frame is frozen and the VCR is paused. If the VCR is paused when this command is issued, the current frame is frozen. The frozen image remains on the output device until an unfreeze command is issued. If neither " input " nor " output " is specified, " output " is assumed. |
| outside | Indicates that the area outside the region specified using the " at " flag is frozen. |

Flags

Can be "**wait**", "**notify**", "**test**", or a combination of these.

Remarks

When used with VCR devices, this command is intended for frame-grabbing cards.

To specify irregular acquisition regions with the "**at**" flag, use a series of freeze and unfreeze commands. Some video-overlay devices limit the complexity of the acquisition region.

This command is supported only if a call to the capability command with the "**can freeze**" flag returns TRUE.

The following command disables video acquisition in a 100-pixel square at the upper left corner of the video buffer:

```
freeze vboard at 0 0 100 100
```


index

The **index** command controls a VCR's on-screen display. VCR devices recognize this command.

MCI Syntax: "index DeviceID Mode Flags"

Parameters

| | |
|----------|---|
| DeviceID | Identifier of an MCI device. This identifier or alias is assigned when the device is opened. |
| Mode | One of the following flags: off Turns off the on-screen display. on Turns on the on-screen display. The item to be displayed is specified by the "index" flag of the set command. |
| Flags | Can be " wait ", " notify ", or " test ". |

info

The **info** command retrieves a hardware description from a device. All MCI devices recognize this command.

MCI Syntax: “info DeviceID Infotype Flags”

Parameters

| | | |
|---------------------|---|-----------------|
| DeviceID | Identifier of an MCI device. This identifier or alias is assigned when the device is opened. | |
| InfoType | Flag that identifies the type of information required. The following table lists device types that recognize the info command and the flags used by each type: | |
| cdaudio | info identity | product |
| | info upc | |
| digitalvideo | audio algorithm | usage |
| | audio quality | version |
| | file | video algorithm |
| | product | video quality |
| | still algorithm | window text |
| | still quality | |
| overlay | file | window text |
| | product | |
| sequencer | copyright file | name product |
| vcr | product | version |
| videodisc | product | |
| waveaudio | file | output |
| | input | product |

The following table lists the flags that can be specified in the **InfoType** parameter and their meanings:

| | |
|------------------------|--|
| audio algorithm | Returns the name of the current audio compression algorithm. |
| audio quality | Returns the name for the current audio quality descriptor. This might return "unknown" if the application has set parameters to specific values that do not correspond to defined qualities. |
| copyright | Retrieves the MIDI file copyright notice from the copyright meta event. |
| file | Retrieves the name of the file used by the compound device. If the device is opened without a file and the load command has not been used, a null string is returned. |
| info identity | Produces a unique identifier for the audio CD currently loaded in the player being queried. |
| info upc | Produces the Universal Product Code (UPC) that is encoded on an audio CD. The UPC is a string of digits. It might not be available for all CDs. |
| input | Retrieves the description of the current input device. Returns "none" if an input device is not set. |
| name | Retrieves the sequence name from the sequence/track name meta event. |
| output | Retrieves the description of the current output device. Returns "none" if an output device is not set. |
| product | Retrieves a description of the device. This information often includes the product name and model. The string length will be 31 characters or fewer. |
| still algorithm | Returns the name of the current still image compression algorithm. |
| still quality | Returns the name for the current still image quality descriptor. This might return "unknown" if the application has set parameters to specific values that do not correspond to defined qualities. |
| usage | Returns a string describing usage restrictions that might be imposed by the owner of the visual or audio data in the workspace. |
| version | Returns the release level of the device driver and hardware. |
| video algorithm | Returns the name of the current video compression algorithm. |

| | |
|---------------|--|
| video quality | Returns the name for the current video quality descriptor. This might return "unknown" if the application has set parameters to specific values that do not correspond to defined qualities. |
| window text | Retrieves the caption of the window used by the device. |
| Flags | Can be " wait ", " notify ", or " test ". |

list

load

mark

monitor

open

paste

pause

play

put

quality

realize

record

reserve

restore

resume

save

seek

set

setaudio

settimecode

settuner

setvideo

signal

spin

status

step

stop

sysinfo

undo

unfreeze

update

where

window

