MilliHelp Quick Reference (Abridged edition)
for PureBasic 4.00 for Windows

v 1.01; www.freewebs.com/millihelp

Window

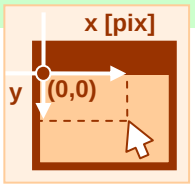
create/free

winID | 0 failed = OpenWindow (#win, x, y [pix]▲, innerW, innerH, title$, flags▲, parentWinID •no parent) ; Activates
CloseWindow (#win)

is#winValid? = IsWindow (#win)
winID = WindowID (#win)
outputID for 2DDRAWING = WindowOutput (#win)

▲ OR'ed flags(•#PB_Window_SystemMenu):
#PB_Window_(SystemMenu|MinimizeGadget|MaximizeGadget|SizeGadget;sizeable border
|Invisible|TitleBar|BorderLess|ScreenCentered;x/y ignored|WindowCentered;on parent|Maximize|Minimize)
▲ If x or y is #PB_Ignore, OS will choose the position

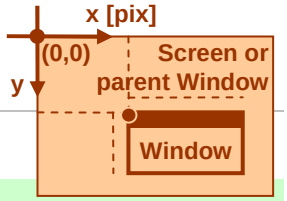
window traits



SetActiveWindow (#win) ; For this prog's wins only. Doesn't auto-bring the prog to foreground
#win = GetActiveWindow ( )

HideWindow (#win, 1 hide | 0 show)
DisableWindow (#win, 1 disable | 0 enable (•)) ; Disables user input
StickyWindow (#win, 1 on top | 0 normal (•)) ; For other prog's wins also
SmartWindowRefresh (#win, 1 enable | 0 disable (•)) ; While resizing

SetWindowState (#win, state▲) state▲ = GetWindowState (#win)
SetWindowTitle (#win, text$) text$ = GetWindowTitle (#win)
SetWindowColor (#win, bgColor▲|–1 reset) bgColor▲|–1 not set = GetWindowColor (#win)



▲ #PB_Window_(Normal|Maximize|Minimize)
▲ ! Use MISC:RGB( )

dimensions & mouse

; In pixels

x = WindowX (#win) w = WindowWidth (#win) x in client area| –1 ms outside win = WindowMouseX▲(#win)
y = WindowY (#win) h = WindowHeight (#win) y in client area| –1 ms outside win = WindowMouseY▲(#win)

ResizeWindow(#win, x|#PB_Ignore, y|#PB_Ignore, w|#PB_Ignore, h|#PB_Ignore to not change)

▲ ! For absolute screen coords see DESKTOP:DesktopMouse(X|Y)( )

event handling

; For windows opened in a thread a separate event loop in necessary

ev▲ | 0 no events = ▲WindowEvent( ) ; Returns immediately. ! Think of using MISC:Delay( ) to not eat up all CPU power
ev▲ = ▲WaitWindowEvent(timeout [millisecs], •waits forever) ; ! Only once per loop

evType▲ = EventType( )
#win = EventWindow ( )
#gdt = EventGadget ( ) ; Also for systray icons
#menuItem = EventMenu ( ) ; Also for toolbar buttons & kbd shortcuts

SetWindowCallback(@myCallback( )▲, #win for which to set, •set for all open windows)
value = EventwParam ( )
value = EventlParam ( )

▲ Native OS events can be catched as well
▲ #PB_Event_((Close|Move|Size|Activate)Window | Repaint|Gadget|Menu;also for toolbar & kbd shortcuts|SysTray)
▲ #PB_EventType_((Left|Right)[Double]Click | Focus|LostFocus | ReturnKey|Change | CloseItem|SizeItem)
▲ Prototype: Procedure myCallback(winID, msg, wParam, lParam)
result = #PB_ProcessPureBasicEvents
. . . . .
ProcedureReturn result
EndProcedure ; Handles all OS events. It's low level

keyboard shortcuts

; Default shorcuts of a window: #PB_Shortcut_Tab, #PB_Shortcut_Tab|#PB_Shortcut_Shift
; Shortcuts generate #PB_Event_Menu event

AddKeyboardShortCut (#win, shortcut▲, #menuItemLikeNumber to be returned by EventMenu( )) ; Adds or replaces
RemoveKeyboardShortCut (#win, shortcut▲|#PB_Shortcut_All remove all)

▲ Any OR'ed combination of #PB_Shortcut_(Shift|Alt|Control) OR'ed with #PB_Shortcut_(one of

A..Z 0..9 F1..F24 Pad0..Pad9 Escape Tab Capital Back Space Return Clear Menu Select Execute
Snapshot Help LeftWindows RightWindows Apps Print Pause Scroll Insert Delete End Home Proir Next Left Up
Right Down Multiply Add Subtract Divide NumLock Separator Decimal)

Gadget

; x,y,w,h [pix]. ! For colors use MISC:RGB( ). If Windows XP uses visual styles, gadget color settings are ignored

create/free

gdtID | 0 failed = KindOfGadget(#gdt ...)
FreeGadget ( ) ; For containing gdt also its gdt list

is#gdtValid? = IsGadget (#gdt)
gdtID = GadgetID (#gdt)

gadget list

; There must be curr. gdt list to create gadgets. ! Use ongoing gadget numbering thru all gadget lists

ok? = CreateGadgetList (winID)
UseGadgetList (winID)

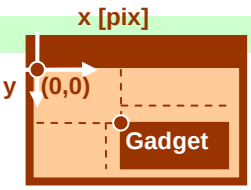
OpenGadgetList (#gdt Container,Panel,ScrollArea, item of Panel: 0..)
CloseGadgetList ( ) ; returns to the prev. gdt list

dimensions

; In pixels from top-left of client area of parent

x = GadgetX(#gdt)
y = GadgetY(#gdt)

w = GadgetWidth (#gdt)
h = GadgetHeight(#gdt)



ResizeGadget (#gdt, x|#PB_Ignore, y|#PB_Ignore, w|#PB_Ignore, h|#PB_Ignore to not change)

gadget traits

SetActiveGadget (#gdt) ; Sets focus
#gdt | –1 no focused gdts = GetActiveGadget( )

HideGadget (#gdt, 1 hide | 0 show (•))
DisableGadget (#gdt, 1 disable | 0 enable (•)) ; Disables user input
GadgetToolTip (#gdt, text$)

type▲ = GadgetType (#gdt)

SetGadgetText	(#gdt, text$ gdt dependent)	text$	= GetGadgetText (#gdt)
SetGadgetState	(#gdt, state gdt dependent)	state	= GetGadgetState (#gdt)
SetGadgetAttribute	(#gdt, attr gdt dependent, value)	value	= GetGadgetAttribute (#gdt, attr)
SetGadgetData	(#gdt, value)	value | 0 not set	= GetGadgetData (#gdt)
SetGadgetColor	(#gdt, colorType▲, color|–1 sys color)	color | 0 not set	= GetGadgetColor (#gdt, colorType▲)

SetGadgetFont (#gdt|#PB_Default gdt creation def. font, fontID|#PB_Default system font)
fontID = GetGadgetFont (#gdt|#PB_Default gdt creation def. font) ; E.g.: for 2DDRAWING:DrawingFont( )

▲ #PB_Gadget_(Front|Back|Line|
TitleFront|TitleBack|GrayText ; for Calendar gdt
)Color

▲ #PB_GadgetType_(one of MDI Image Text Frame3D ButtonImage Button CheckBox OptionHyperLink
String Editor IPAddress ComboBox ListView Tree ListIcon Explorer(Combo|List|Tree)
Container Panel ScrolArea Splitter ScrollBar TrackBar Spin ProgressBar Web Calendar Date)

gadget items

#kidWin for MDI only = AddGadgetItem▲ (#gdt, pos 0..| –1 to append, text$, imgID 16x16 *.ico, flags)
RemoveGadgetItem(#gdt, pos 0..)

n = CountGadgetItems (#gdt)
ClearGadgetItemList (#gdt)

gadgets with items: user-defined images for items are allowed

SetGadgetItemText	(#gdt, item 0..|–1 header , text$ gdt dep., col▲0..)	text$	= Get...(#gdt, item 0.. , col▲0..)
SetGadgetItemState	(#gdt, item 0.. , state gdt dep.)	state	= Get...(#gdt, item 0..)
SetGadgetItemAttribute	(#gdt, item 0.. , attr gdt dep., value, col▲0..)	value	= Get...(#gdt, item 0.. , attr, col▲0..)
SetGadgetItemData	(#gdt, item 0.. , value)	0 not set | value	= Get...(#gdt, item 0..)

SetGadgetItemColor (#gdt, item 0..|–1 to all items, colorType▲, color|–1 reset, col 0..|–1 to all cols)
color | –1 not set = GetGadgetItemColor (#gdt, item 0.., colorType▲, col 0..)

itemID = GadgetItemID(#gdt, itemPos 0..) ; For Tree gdt

▲ If MDI gdt: pos is #kidWin (if pos=#PB_Any, #kidWin is returned)
flags – same as for OpenWindow( ) except #PB_Window_(BorderLess|(Screen|Window)Centered)
If Tree gdt: flags is sublevel (0..) & it's mandatory
▲ For ListIcon, ExplorerList gdts. –1 means header
▲ #PB_Gadget_(Front|Back)Color

gadget columns

; For ListIcon, ExplorerList gadgets

AddGadgetColumn (#gdt, pos 0.. , title$▲, width [pix])
RemoveGadgetColumn (#gdt, pos 0..)

ChangeListIconGadgetDisplay(#gdt, mode #PB_ListIcon_(LargeIcon|SmallIcon|List|Report))

▲ For ExplorerList gdt "auto-columns": #PB_Explorer_(Name|Size|Type|Attributes|Created|Modified|Access)

Gadget (2)

individual gadget constructors

gdtID | 0 failed = MDIGadget (#gdt, x, y, w, h, subMenu 0.. , #firstMdiMenuItem, flags OR'ed (•0) #PB_MDI_(▲))

▲ AutoSize|BorderLess|NoScrollBars

gdtID | 0 failed = ImageGadget (#gdt, x, y, w, h, imgID, flags OR'ed (•0): #PB_Image_Border)
gdtID | 0 failed = TextGadget (#gdt, x, y, w, h, text$, flags OR'ed (•0): #PB_Text_((Center|Right) | Border))
gdtID | 0 failed = Frame3DGadget (#gdt, x, y, w, h, text$, flags #PB_Frame3D_(Single|Double|Flat))

gdtID | 0 failed = ButtonImageGadget(#gdt, x, y, w, h, imgID)
gdtID | 0 failed = ButtonGadget (#gdt, x, y, w, h, text$, flags OR'ed (•0): #PB_Button_(▲))
gdtID | 0 failed = CheckBoxGadget (#gdt, x, y, w, h, text$, flags #PB_CheckBox_(Right|Center))
gdtID | 0 failed = OptionGadget (#gdt, x, y, w, h, text$)
gdtID | 0 failed = HyperGadget (#gdt, x, y, w, h, text$, color)

▲ (Right|Left) |Default|MultiLine|Toggle

gdtID | 0 failed = StringGadget (#gdt, x, y, w, h, text$, flags OR'ed (•0): #PB_String_(▲))
gdtID | 0 failed = EditorGadget (#gdt, x, y, w, h)
gdtID | 0 failed = IPAddressGadget (#gdt, x, y, w, h)

▲ Numeric|Password|ReadOnly| (Lower|Upper)Case |BorderLess

gdtID | 0 failed = ComboBoxGadget (#gdt, x, y, w, h, flags OR'ed(•0): #PB_ComboBox_(Editable| (Lower|Upper)Case))
gdtID | 0 failed = ListViewGadget (#gdt, x, y, w, h)
gdtID | 0 failed = TreeGadget (#gdt, x, y, w, h, flags OR'ed (•0): #PB_Tree_(▲))
gdtID | 0 failed = ListIconGadget (#gdt, x, y, w, h, text$, textW, flags OR'ed (•0): #PB_ListIcon_(▲))

▲ AlwaysShowSelection|NoLines|NoButtons|CheckBoxes
▲ CheckBoxes|MultiSelect|GridLines|FullRowSelect|HeaderDragDrop|AlwaysShowSelection

gdtID | 0 failed = ExplorerComboGadget (#gdt, x, y, w, h, dirName$, flags OR'ed (•0): #PB_Explorer_(▲))
gdtID | 0 failed = ExplorerListGadget (#gdt, x, y, w, h, dirName$, flags OR'ed (•0): #PB_Explorer_(▲))
gdtID | 0 failed = ExplorerTreeGadget (#gdt, x, y, w, h, dirName$, flags OR'ed (•0): #PB_Explorer_(▲))

▲ DrivesOnly|Editable|NoMyDocuments
▲ BorderLess|AlwaysShowSelection|MultiSelect|GridLines|HeaderDragDrop|FullRowSelect|
NoFiles|NoFolders|NoParentFolder|NoDirectoryChange|NoDriveRequester|NoSort|NoMyDocuments|AutoSort
▲ BorderLess|AlwaysShowSelection|NoLines|NoButtones| NoFiles|NoDriveRequester|NoMyDocuments|AutoSort

gdtID | 0 failed = ContainerGadget (#gdt, x, y, w, h, flags #PB_Container_(BorderLess(•)|Flat|Raised|Single|Double))
gdtID | 0 failed = PanelGadget (#gdt, x, y, w, h)
gdtID | 0 failed = ScrollAreaGadget (#gdt, x, y, w, h, scrW, scrH, scrStep, flags OR'ed: #PB_ScrollArea_(▲))
gdtID | 0 failed = SplitterGadget (#gdt, x, y, w, h, #gdt1, #gdt2, flags OR'ed: #PB_Splitter_(▲))

▲ (Flat|Raised|Single|BorederLess) |Center
▲ Vertical|Separator| (First|Second)Fixed

gdtID | 0 failed = ScrollBarGadget (#gdt, x, y, w, h, min, max, pgLen, flags▲)
gdtID | 0 failed = TrackBarGadget (#gdt, x, y, w, h, min, max, flags▲)
gdtID | 0 failed = SpinGadget (#gdt, x, y, w, h, min, max)
gdtID | 0 failed = ProgressBarGadget (#gdt, x, y, w, h, min, max, flags OR'ed(•0): #PB_ProgressBar_(Smooth|Vertical))

▲ OR'ed (•0): #PB_ScrollBar_Vertical
▲ OR'ed (•0): #PB_TrackBar_(Ticks|Vertical)

gdtID | 0 failed = WebGadget (#gdt, x, y, w, h, URL$, flags OR'ed (•0): #PB_Web_Mozilla)

gdtID | 0 failed = CalendarGadget (#gdt, x, y, w, h, date •curr., flags OR'ed (•0): #PB_Calendar_BorderLess)
gdtID | 0 failed = DateGadget (#gdt, x, y, w, h, fmt$ •""|"" default fmt, date •curr| 0 curr., flags▲)

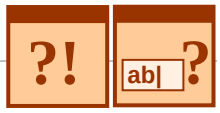
▲ OR'ed (•0): #PB_Date_(UpDown|CheckBox)

Requester

; A requester (known also as a dialog) suspends program execution until the user closes it
; Win API constants can also be used
; ! See also: PRINTER:PrintRequester( ), DATABASE:DatabaseRequester( )

message requesters

btnPressed▲ = MessageRequester(title$, message$! use Chr(13) to break lines, buttons▲)
input$ | "" cancelled = InputRequester (title$, message$, initInputText$)



▲ #PB_MessageRequester_(Yes|No|Cancel)
▲ #PB_MessageRequester_(Ok•|YesNo|YesNoCancel)

filesystem requesters

fullPath$ "\"-terminated | "" cancelled = PathRequester(title$▲, defaultPath$▲ terminating "\" is optional)

fullFname$▲ | "" cancelled = SaveFileRequester (title$▲, defaultPathAndFname$▲, pattern$▲, defPatternPos 0..)
fullFname$▲ | "" cancelled = OpenFileRequester(title$▲, defaultPathAndFname$▲, pattern$▲, defPatternPos 0.., flags▲)

fullFname$▲ | "" no more = NextSelectedFileName( ) ; ! Call repeatedly after OpenFileRequester( ) w/ _Multiselection on
patternPos 0.. | –1 cancelled or no pattern = SelectedFilePattern( ) ; ! Call after (Save|Open)FileRequester( )

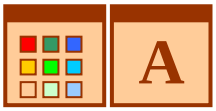
▲ To split into parts: MISC:GetPathPart( ),GetFilePart( ),GetExtensionPart( )
▲ If "", then system default values are used
▲ E.g.: "Text files | *.txt | Music files | *.mp3; *.mid"
pattern 0 pattern 1
▲ OR'ed flags (•0): #PB_Requester_MultiSelection

color & font requesters

color▲ | –1 cancelled = ColorRequester (defaultColor▲ •black)

1 selected | 0 cancelled = FontRequester (initFontName$ | "" blank, defaultFontSize | 0 blank, flags▲, defaultColor▲)

name$ = SelectedFontName ( ) ; ! These 4 commands are to be called after FontRequester( )
color = SelectedFontColor ( )
size = SelectedFontSize ( )
style = SelectedFontStyle ( ) ; OR'ed flags (•0): #PB_Font_(Bold|Italic)



▲ ! 24 bits. See color handling commands: MISC:Red( ),Green( ),Blue( ),RGB( )
▲ OR'ed flags (•0): #PB_FontRequester_Effects ; enable font effects options in requester

Console

; Only 1 console in a program at a time
; If a program has a console, but no GUI (Windows/Gadgets), set (Compiler Options | Executable Format) to "Console"
; A console-program isn't an MS-DOS program, it requires Windows
; In text mode: stdin/stdout streams (used by Input( ), Print[N]( ), [Read|Write]ConsoleData( )) can be redirected
(E.g.: in Windows command console, "myprog1.exe | myprog2.exe > file3.txt")

open/close

ok? = OpenConsole ( ) ; In text mode. ! Call before other CONSOLE commands
CloseConsole ( ) ; Auto-closed on program exit

output

Print (text$) ; Both wrap text$ is text mode only
PrintN (text$) ; Appends NEWLINE char to the output

ConsoleError(msg$); To strerr. Adds NEWLINE. ! See PROCESS:ReadProgramError( )

ClearConsole ( ) ; By curr. back color, sets caret to (0,0). Graph. mode only
ConsoleLocate (x 0.. [chars], y 0.. [chars]) ; Graph. mode only

input

str$|#PB_Input_Eof text mode only = Input( ) ; Waits for ENTER-key. In graph. mode line len is limited by right edge

ASCIIChar$ | "" no ASCII keys being pressed = Inkey ( ) ; The most resently pressed & unreleased ASCII key
ASCIICode | ExtCode for non-ASCII keys = RawKey( ) ; The result of the last call to Inkey( ) for non-ASCII keys

console traits

ConsoleTitle (title$)
ConsoleCursor (height 0:invisible, 1:underline (•), 5:mid-height, 10:full height)
ConsoleColor (textColor 0..15, backColor 0..15)

EnableGraphicalConsole▲ (state 1: graphical mode, 0: text mode)

▲ Only in text mode: redirection, [Read|Write]ConsoleData( ), wrapping Print( )'ed text
Only in graphical mode: ClearConsole( ) & ConsoleLocate( ), truncating Print() 'ed text

text-mode raw input/output for redirection

bytesWritten = WriteConsoleData(*memBuf, size)
bytesRead | 0 end of file = ReadConsoleData(*memBuf, size) ; Waits for size bytes to be read or end-of-file

Menu

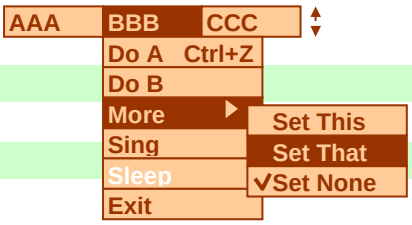
create/free

menuID | 0 failed = CreateMenu (#menu, winID for which to create menu)
menuID | 0 failed = CreateMenu (#menu)
FreeMenu (#menu)

is#menuValid? = IsMenu (#menu)
menuID = MenuID (#menu)

show/hide

HideMenu (#menu, 1 hide | 0 show)
DisplayPopupMenu (#menu, winID some valid win, x, y from screen's top-left)



height

h of menu title bar [pix] = MenuHeight( )

menu building

; Items are added to the currently opened menu/submenu

MenuTitle (title$▲) ; Adds menu on the menu bar

OpenSubMenu (text$▲)
CloseSubMenu ( )

MenuItem (#menuItem, text$▲▲)
MenuBar ( ) ; Adds a horiz. separator

▲ "&" makes a letter active (e.g.: "E&xit" gives "Exit")
▲ TAB char is used to add a shortcut (e.g.: "Do this"+Chr(9)+"Ctrl+Z"). ! Activate it by WINDOW:AddKeyboardShortcut( )

menu item traits

DisableMenuItem (#menu, #menuItem, 1 disable | 0 enable)

`res = SetMenuItemState (#menu, #menuItem, 1 display check mark | 0 remove it)
1 checked | 0 not checked = GetMenuItemState (#menu, #menuItem)

SetMenuItemText (#menu, #menuItem, text$)
text$ = GetMenuItemText (#menu, #menuItem)
SetMenuTitleText (#menu, title 0.. , text$)
text$ = GetMenuTitleText (#menu, title 0..)

menu events

; On menu event
WINDOW:[Wait]WindowEvent( ) returns #PB_Event_Menu
WINDOW:EventMenu( ) returns appropriate #menuItem.
; ! See also TOOLBAR – toolbar buttons raise events as if they were menu items

ToolBar

create/free

toolBarID | 0 failed = CreateToolBar(#toolBar, winID for which to create toolbar)
FreeToolBar (#toolBar)

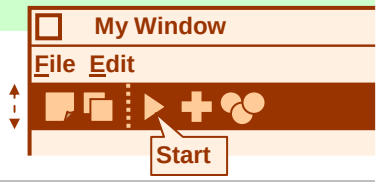
is#toolBarValid? = IsToolBar (#toolBar)
toolBarID = ToolBarID(#toolBar)

building buttons

; Button are added on the currently created tool bar

ToolBarStandardButton (#toolBarBtn, btnIcon▲, mode #PB_ToolBar_(Normal•|Toggle))
ToolBarImageButton (#toolBarBtn, imgID▲! use *.ico format for transparent)

ToolBarSeparator( )



▲ #PB_ToolBarIcon_(New|Open|Save|Print|Find|Replace| Cut|Copy|Paste|Undo|Redo| Delete|Properties|Help)
▲ ! See IMAGE:ImageID( ) & image creation commands

height

h [pix] = ToolBarHeight (#toolBar)

button traits

DisableToolBarButton (#toolBar, #toolBarBtn, 1 disable | 0 enable)

SetToolBarButtonState ▲ (#toolBar, #toolBarBtn, 1 pushed | 0 released)
1 pushed | 0 released = GetToolBarButtonState ▲ (#toolBar, #toolBarBtn)

ToolBarToolTip (#toolBar, #toolBarBtn, text$)

▲ For mode=#PB_ToolBar_Toggle

toolbar events

; Toolbar buttons raise events as if they were menu items. It's a good idea for them to have the same identifying #number
; On toolbar event
WINDOW:[Wait]WindowEvent( ) returns #PB_Event_Menu (like for menus)
WINDOW:EventMenu( ) returns appropriate #toolBarBtn.

StatusBar

create/free

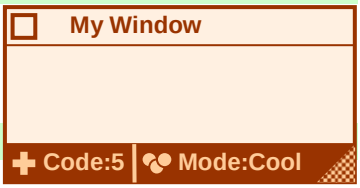
sbarID | 0 failed = CreateStatusBar (#sbar, winID for which to create status bar)
FreeStatusBar (#sbar)

is#sbarValid? = IsStatusBar(#sbar)
sbarID = StatusBarID(#sbar)

building fields

AddStatusBarField(width [pix]) ; Adds fields from left to right

StatusBarText(#sbar, field 0.. , text$, appearance▲)
StatusBarIcon(#sbar, field 0.. , imgID▲16x16 *.ico format)



height

h [pix] = StatusBarHeight(#sbar)

▲ OR'ed flags (●0): #PB_StatusBar_(Raised|BorderLess| (Center|Right))
▲ ! See IMAGE:ImageID( ) & image creation commands

SysTray

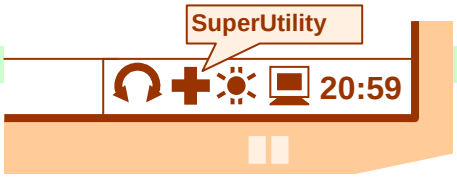
add/remove icons

sysTrayIconID | 0 failed = AddSysTrayIcon (#sysTrayIcon, winID some valid win, imgID▲ *.ico format)
ChangeSysTrayIcon (#sysTrayIcon, imgID▲ *.ico format)
RemoveSysTrayIcon (#sysTrayIcon) ; All icons are auto-removed on program exit

is#sysTrayIconValid? = IsSysTrayIcon (#sysTrayIcon)

icon tooltip

SysTrayIconToolTip (#sysTrayIcon, text$)



▲ ! See IMAGE:ImageID( ) & image creation commands

systray events

```
; On systray event
  WINDOW:[Wait]WindowEvent( )  returns #PB_Event_SysTray
  WINDOW:EventGadget( )        returns appropriate #sysTrayIcon
  WINDOW:EventType( )          returns the event type
```

Desktop

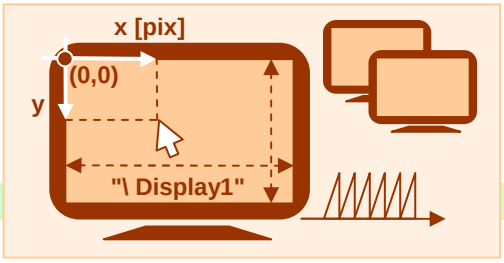
; Multiple screens can be connected to your computer

grab info

numberOfDesktops | 0 failed = ExamineDesktops( ) ; ! Call before desktop-traits commands

desktop traits: name, frequency, dimensions

name$ | "" not found = DesktopName (#desktop 0 (primary) ..)
freq [Herz] | 0 default freq. = DesktopFrequency (#desktop 0 (primary) ..)
w [pix] = DesktopWidth (#desktop 0 (primary) ..)
h [pix] = DesktopHeight (#desktop 0 (primary) ..)
bitsPerPixel 1,2,4,8,15,16,24,32 = DesktopDepth (#desktop 0 (primary) ..)



mouse

x [pix] = DesktopMouseX( )
y [pix] = DesktopMouseY( )

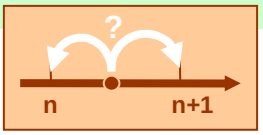
Math

; If you pass a float (.f) argument to a command, the return type is also float. If you pass double (.d), it returns double
; ! See also: MISC:Random( ),RandomSeed( )

rounding

y.f|d = Round (x.f|d, mode 0: round towards –, 1: round towards +)

n.l = Int (x.f|d) ; Rounds towards zero
n.q = IntQ (x.f|d) ; Rounds towards zero



absolute value

y.f|d = Abs(x.f|d) ; ! Don't pass an integer, convert to a float explicitly (e.g.: multiply it by 1.0)

(un)powers

y.f|d = Pow (base.f|d, exp.f|d)

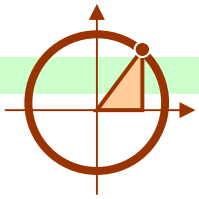
y.f|d = Log (x.f|d >0) ; Natural log
y.f|d = Log10 (x.f|d >0)

y.f|d = Sqr(x.f|d >=0) ; Square root

trigonometry

y.f|d [–1;1] = Sin (x.f|d [radians])
y.f|d [–1;1] = Cos (x.f|d [radians])
y.f|d = Tan (x.f|d [radians])

x.f|d [–pi/2; pi/2], [radians] = ASin (y.f|d [–1;1])
x.f|d [0; pi], [radians] = ACos (y.f|d [–1;1])
x.f|d (–pi/2; pi/2), [radians] = ATan (y.f|d)



String

; ! See also MEMORY commands for fast but manual text handling
; Lengths & positions/indexes are in chars, if not specified
; STRING commands don't change input strings (exception: ReplaceString( ) with "In-place" flag on)

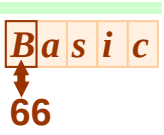
length

len [chars] = Len (str$)
len [bytes] = StringByteLength (str$, format▲)

▲ #PB_(Ascii|UTF8|Unicode) – overrides the compile mode for his command,
• #PB_(Ascii|Unicode) depending on the compile mode

char codes

ascii of 1st char, 0..255 =Asc(str$)
char$ =Chr (ascii 0..255)



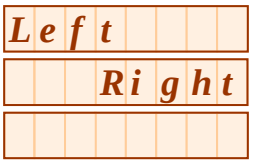
char case

res$ =UCase (str$) ; Accent letters – ok
res$ =LCase (str$) ; -/-

spaces

res$ = LTrim (str$)
res$ = RTrim (str$)
res$ = Trim (str$)

res$ = LSet (str$, len, fillChar$ • " ") ; Truncates end of str$, if too long
res$ = RSet (str$, len, fillChar$ • " ") ; -/-
res$ = Space(len)

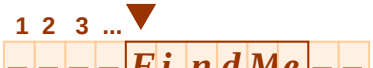


substrings

res$ = Left (str$, len)
res$ = Right (str$, len)
res$ = Mid (str$, startPos 1.. , len)

field$ = StringField(str$, fieldIdx 1.. , singleCharDelimiter$)

pos 1.. | 0 not found = FindString (str$, strToFind$, startPos 1..)
numOfOccurences = CountString (str$, strToCount$)
res$ = RemoveString (str$, strToRemove$, mode OR'ed flags (•0): 1-case insensitive)
res$ | ignore if "In-place" flag = ReplaceString (str$, strToFind$, strToReplaceWith$, mode▲, startPos 1(•) ..)



▲ OR'ed flags (•0):
1: Case-insensitive,
2: In-place replacement (NOTE: it changes str$; strToFind$ & strToReplaceWith$ must be of same len; ! ignore the result)

string to number

res$ = StrF (value.f , decPlaces)
res$ = StrD (value.d, decPlaces)

res$ = Str (value.l)
res$ = StrQ (value.q)

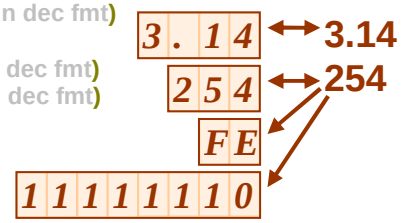
res$ = StrU (value, type (#Byte|#Word|#Long|#Quad))

res$ = Hex (value) res$ = HexQ (value.q)
res$ = Bin (value) res$ = BinQ (value.q)

number to string

res.f = ValF (str$ float in decimal format)
res.d = ValD (str$ double in dec fmt)

res.l = Val (str$ int in dec fmt)
res.q = ValQ (str$ quad in dec fmt)



LinkedList

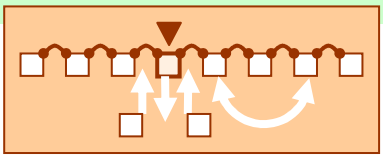
; ( ) after list name are mandatory
; If comparing to arrays, linked lists are faster for element insert/remove, but slower for element access

list declaration

NewList lst . ElemType▲usually a structure ( )

▲ PB adds two fields to ElemType:
```
Structure ElemType
  *Next     . ElemType ; ! Don't change
  *Previous . ElemType ; ! Don't change
  ; user-defined data here
EndStructure
```

modifying elements



lst( ) = ... ; Set the value of the curr. element
... = lst( ) ; Get the value of the curr. element

*newElemPtr becomes curr. | 0 failed = AddElement (lst( )) ; After the curr. elem
*newElemPtr becomes curr. | 0 failed = InsertElement (lst( )) ; Before the curr. elem

▲DeleteElement (lst( ), ifDeletingTheFirstElem▲) ; The prev. elem becomes curr.
▲ClearList (lst( ))
SwapElements (lst( ), *elem1, *elem2 both got as @lst( ) & point to valid elems)

▲ 0(•): no new curr., 1: set 2nd as new curr. (if exists)
▲ Deleting elements just excludes them from the list, but doesn't free any resources allocated for them

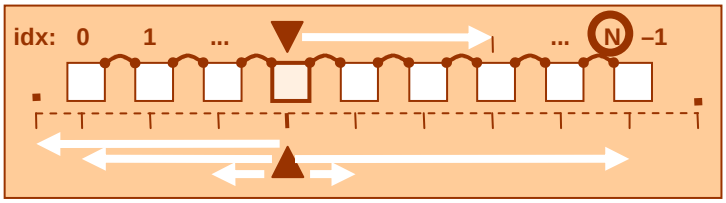
moving current position

; There can be no curr. elem: list is empty, curr. pos is before 1st or after last elem

numberOfElems = CountList(lst( )) ; Very fast, no iteration involved
–1 no curr elem | currElemIdx 0.. = ListIndex (lst( )) ; Very fast, no iteration involved

SelectElement(lst( ), newCurrElemIdx 0.. , only valid idx) ; Iterates thru elems, not fast
ChangeCurrentElement(lst( ), *newCurrElemPtr got as @lst( ), only valid elemptr)

ResetList(lst( )) ; No new curr.



*elemPtr | 0 list is empty = FirstElement (lst( ))
*elemPtr | 0 list is empty = LastElement (lst( ))

*elemPtr | 0 if at the end = NextElement (lst( ))
*elemPtr | 0 if at the end = PreviousElement (lst( ))

list iteration

```
ForEach lst( )
  ... lst( ) ...
Next [ lst( ) ]
```

Cipher

encoding/decoding

; Base64 algorithm is good for 7-bits ASCII data with char codes 32..127

Base64Encoder(*inputBuf, inputLen [bytes], *outputBuf, outputLen▲[bytes])
Base64Decoder(*inputBuf, inputLen [bytes], *outputBuf, outputLen▲[bytes])

"PureBasic"
Base64 "UHVyZUJhc"

▲ Up to 33% longer than input (! reserve enough space), but 64 bytes minimum
▲ Down to 33% shorter than input, but 64 bytes minimum

fingerprints

; Hashing message

fingerprint$13 chars = DESFingerprint▲(password$▲, key$▲) ; Resistant. Good for hashing passwords
fingerprint 32 bits = CRC32Fingerprint(*buf, len [bytes]) ; Not crack resistant. Good for error detection. Very fast
fingerprint$ 32 chars = MD5Fingerprint (*buf, len [bytes]) ; Resistant. Good for long data
fingerprint$ 32 chars | "" not found = MD5FileFingerprint(fname$) ; -/-

▲ Up to 8 first chars are taken into account
▲ Also known as 'salt' in Unix world. Often, consists of 2 chars
▲ Based on opensource crypt( ) command

"PureBasic"	
DES	"AAPb0Ym0OOhOo" (key$="AA")
CRC32	4CBF388D (in hex)
MD5	"ed50deb5bb795508b8a5c8e50dafa954"

Memory

; *memIDs are addresses of memories allocated by [Re]AllocatMemory( )
; All remaining allocated memory is freed automatically on program exit
; String MEMORY commands are optimized for very fast handling of textual data
; To pass a string to *str parameter use @mystr$ or @"ABC"

▲ #PB_(Ascii|UTF8|Unicode) – overrides the compile mode for his command,
• #PB_(Ascii|Unicode) depending on the compile mode

allocate/free

```
    *memID | 0 failed                          = AllocateMemory   (              size [bytes]) ; Initializes the mem with 0
*newMemID | 0 failed (old mem stays intact)= ReAllocateMemory (*oldMemID▲,size [bytes]) ; Inits extra mem with 0
                                              FreeMemory       (memID | –1 free all memories)
```

▲ If 0, just allocates. Valid *memID is returned by [Re]AllocateMemory( )

memory size

```
len [bytes] = MemorySize           (*memID returned by [Re]AllocateMemory( ))
len [chars] = MemoryStringLength   (*str null-terminated, flags▲)
```

compare

```
      yesNo  = CompareMemory       (*mem1,  *mem2,                                  len [bytes])
–1 < | 0 = | 1 > = CompareMemoryString(*str1,   *str2,   isCaseInsensitive? • 0, len [chars] | –1▲(•),  flags▲)
```

▲ • Considers *str1 & *str2 as null-terminated

copy

```
                MoveMemory  (*fromMem, *toMem, len [bytes]) ; Memories may     overlap   (Slower)
                CopyMemory  (*fromMem, *toMem, len [bytes]) ; Memories may not overlap   (Faster)

*nextDestMem = CopyMemoryString▲(*str null-terminated | str$, @Var_*destMem▲• previous value)
```

▲ Address of a variable (Var) whose value is *destMem
! For the first time you call CopyMemoryString( ), provide explicit argument
▲ Terminating null-char is copied as well (! Reserve space for it)
Modifies the value of Var (makes it point at the null-char in the end), and returns it. Good for cascading copying
You can modify the value of Var between calls, changing the curr. *destMem

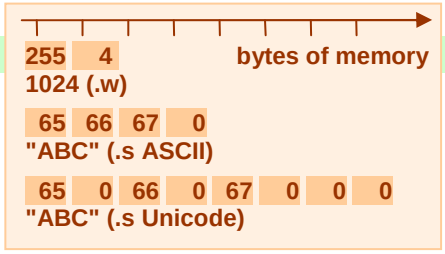
data types in memory

```
number.b = PeekB (*mem)      PokeB (*mem, number.b ) ; 1 byte
number.w = PeekW (*mem)      PokeW (*mem, number.w ) ; 2 bytes
number.l = PeekL (*mem)      PokeL (*mem, number.l ) ; 4 bytes
number.q = PeekQ (*mem)      PokeQ (*mem, number.q ) ; 8 bytes

number.f = PeekF (*mem)      PokeF (*mem, number.f ) ; 4 bytes
number.d = PeekD (*mem)      PokeD (*mem, number.d ) ; 8 bytes

number.c = PeekC (*mem)      PokeC (*mem, number.c ) ; 1 (or 2) bytes in ASCII (Unicode) compile mode

text.s   = PeekS (*mem,        ▲len [chars] | –1 auto (•), flags▲) ; If len is default, then peeks until null-char
           PokeS (*mem, text.s, ▲len [chars] | –1 auto (•), flags▲) ; Appends null-char (! Reserve space for it)
```



▲ Not counting terminating null-char

Sort

; For sorting, null elements are LESS THAN non-null
; ( ) are mandatory

▲ • the whole array/list is sorted
▲ OR'ed flags (•0): 1-switch from ascending order to descending, 2-case insensitive

array/list of basic types

; Basic types: byte (.b), word (.w), long (.l), quad (.q), float (.f), double (.d), character (.c), string (.s)

```
SortArray (arrayName( ), options▲,                                  ▲startIdx 0.. , endIdx inclusive, 0..)
SortList  (  listName( ), options▲,                                 ▲startIdx 0.. , endIdx inclusive, 0..)
```

array/list of Structure

```
SortStructuredArray (arrayName( ), options▲, OffsetOf(Structure\Field), type▲, ▲startIdx 0.. , endIdx inclusive, 0..)
SortStructuredList  (  listName( ), options▲, OffsetOf(Structure\Field), type▲, ▲startIdx 0.. , endIdx inclusive, 0..)
```

▲ #PB_Sort_(Byte|Word|Long|Quad|Float|Double |Character|String)

Date

; Date values are in seconds since 1 Jan 1970 (as in Unix)
; Valid range: 1 Jan 1970 .. 31 Dec 2034

contructors

```
        nowMoment = Date( )
–1 out of range | date  = Date(year 1970..2034, month 1..12, day 1..31, hour 0..23, minute 0..59, second 0..59)
```

components of date

```
Y 1970..2034 = Year    (date)      h  0..23 = Hour      (date)
M 1..12      = Month   (date)      m  0..59 = Minute    (date)      doy 1..366                 = DayOfYear   (date)
D  1..31     = Day     (date)      s  0..59 = Second    (date)      dow 0:Sunday..6:Saturday   = DayOfWeek  (date)
```

adding dates

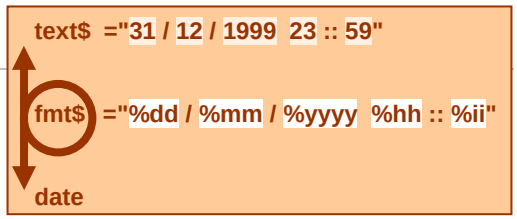
```
newDate = AddDate(oldDate, whatToAdd▲, value valid for selected whatToAdd)
```

▲ #PB_date_(Year|Month|Day|Hour|Minute|Second)

conversion between dates & strings

```
text$          = FormatDate  (format$▲, date)
date | –1 failed  = ParseDate   (format$▲, text$ fields delimited by any chars)
```

▲ Formatting fields are: %yyyy, %yy, %mm, %dd, %hh, %ii, %ss



Packer

; KCalG1 algorithm is used: compression is a bit slow, but compression size is small & decompression is very fast
; All lengths/sizes/positions are in bytes

in-memory compression

```
   packedLen | 0 failed =    PackMemory  (fromMemAddr, toMemAddr▲, fromLen, compessionLevel▲)
unpackedLen | 0 failed = UnPackMemory  (fromMemAddr, toMemAddr▲, fromLen, compessionLevel▲)
```

▲ ! Reserve fromLen+8 bytes
▲ ! Reserve enough(?) bytes
▲ 0:fastest .. 9:most effecient, ●?

open/close

```
ok? = CreatePack (fname$)  ; Empty pack
ok? = OpenPack   (fname$)
```

`ok? = ClosePack( ) ; Saves all opened packs & frees their memories. Done automatically on program exit

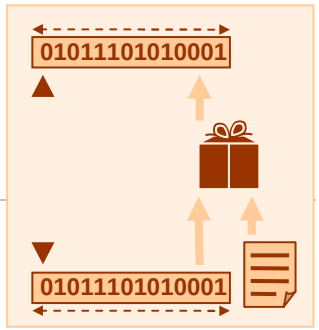
pack reading/writing

; Each packed file may contain the contents of memory & of other files
; These commands work with the most recently created/opened pack

```
memAddr ! don't free this mem | 0 no more files = NextPackFile( )
                                          size = PackFileSize ( ) ; ! Use after NextPackFile( )

`ok? =AddPackFile       (fname$,          compressionLevel▲)
`ok? =AddPackMemory (memAddr, len, compressionLevel▲)
```

▲ 0:fastest .. 9:most effecient, ●?



compression progress callback

```
PackerCallback(@myCompressionProgressCallback( )▲) ; For (un)packing operations
```

▲ Prototype: procedure myCompressionProgressCallback(currFromBufPos 0.. , currToBufPos 0..)
compression to be continued, if it returns 1
. . . cancelled . . . 0

Printer

; 2DDRAWING lib is used to form output for printing

printer selection

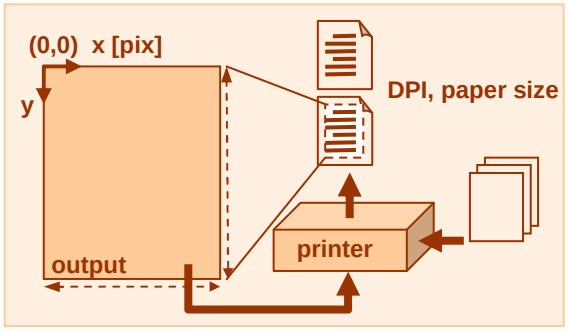
; ! Select a printer by any of these two commands before other PRINTER commands

```
printerSelected? no printers or user cancelled selection if 0  = PrintRequester( ) ; ! See also REQUESTER lib
          ok? unavailable if 0                                 = DefaultPrinter  ( )
```

start/stop

; ! Call 2DDRAWING commands to the PrinterOutput between these two

```
ok? = StartPrinting(jobName$ shown in the printer spooler)
      {
      StopPrinting▲( ) ; Sends the data to the printer
```



output

```
outputID = PrinterOutput( ) ; For 2DDRAWING:StartDrawing( )
```

page control

```
w ▲ of printable area of a page [pix] = PrinterPageHeight( )
h ▲ of printable area of a page [pix] = PrinterPageHeight( )

NewPrinterPage( ) ; Sends data for previous page to the printer
```

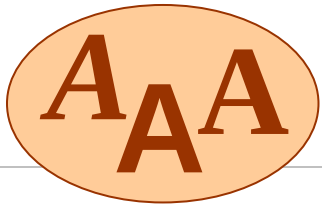
▲ Proportional to dpi of the printer, normally 300 dpi

Font

; ! See also REQUESTER:FontRequester( )
; Using color with fonts isn't still supported

```
fontID▲| 0 failed =  LoadFont▲ (#font, name$ e.g.:"Arial" , height [pix], flags▲)
                    FreeFont    (#font)

        is#fontValid?  = IsFont    (#font)
              fontID▲  = FontID    (#font)
```



▲ OR'ed flags (●0): #PB_Font_(Bold|Italic|Underline|StrikeOut|HighQuality)
▲ If no font exists with given attributes, the closest available font will be loaded
▲ For use with 2DDRAWING:DrawingFont( ), GADGET:Get-/SetGadgetFont( )

2DDrawing

; Dimensions & coords are in pixels
; ! Use MISC:RGB( ) to define color values
; On MS Windows: graphics painted with WINDOW:WindowOutput( ) on a window usually should be redrawn in the #PB_Event_Repaint event handler
; If you draw with IMAGE: ImageOutput( ) on an image of ImageGadget, redrawing is automatic. But after drawing changes, GADGET:SetGadgetState( ) is necessary

start/stop

; ! Call other 2DDRAWING commands between these two

```
hdlOfDC | 0 failed = StartDrawing(outputID▲) ; sets current front color to black (0,0,0), back color to white (255,255,255)
                    {
                    StopDrawing( )
```

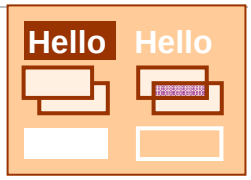
▲ Returned by any of: WINDOW:WindowOutput( ), IMAGE:ImageOutput( ), PRINTER:PrinterOutput( ), SPRITE:SpriteOutput( ), SPRITE:ScreenOutput( ), TEXTURE:TextureOutput( )

current settings

```
FrontColor  (color) ; For text, for primitives outline & fill
BackColor   (color) ; For text background
DrawingFont (fontID | #PB_Default default sys. font)
DrawingMode (mode▲)
```

▲ OR'ed flags (• 0 = #PB_2DDrawing_Default):

```
#PB_2DDrawing_Transparent ; transparent text background,
#PB_2DDrawing_Xor         ; all graphics XOR'ed with background,
#PB_2DDrawing_Outlined    ; switches from filling shapes to outlining them
```



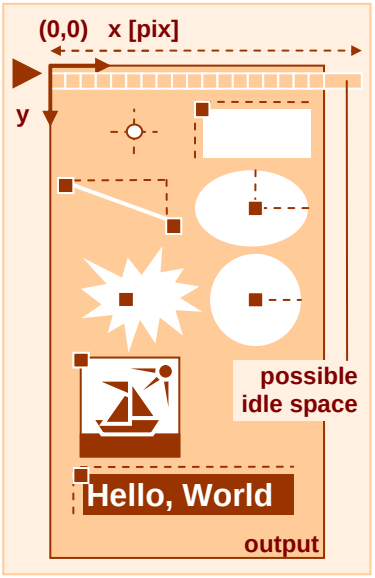
drawing primitives

```
color = Point (x, y) ; ! Use MISC:Red( ),Green( ),Blue( ) to analyze color
        Plot  (x, y,          color •curr. front color)

Line    (x,  y,  dx, dy,      color •curr. front color)
LineXY  (x1, y1, x2, y2,      color •curr. front color)

Box     (x,  y, dx, dy,       color •curr. front color)
Ellipse (x0, y0, xr, yr,      color •curr. front color)
Circle  (x0, y0,  r,          color •curr. front color)

FillArea(x, y, borderColor,   color •curr. front color |–1 color of (x,y)-pixel )
```



drawing images

```
DrawImage      (imgID, x, y, newW, newH original image preserves its size)
DrawAlphaImage(imgID, x, y) ; For 32-bit *.png & *.tiff images
```

drawing text

```
nextXPos [pix] = DrawText(x, y, t$, frontColor •curr. fcolor, backColor •curr. bcolor)

w  [pix] = TextWidth  (t$) ; Both depend on the curr. output & font
h  [pix] = TextHeight (t$) ;
```

direct access to video buffer

; For Screen- or SpriteOutput

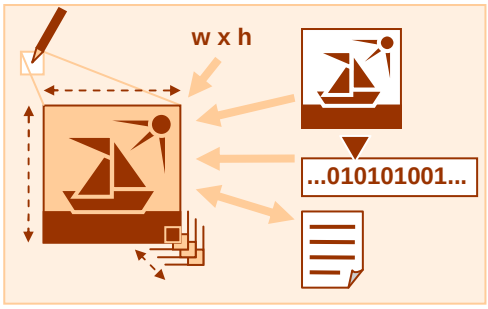
```
0 unavailable | memAddr   = DrawingBuffer( ) ; ! Call it each time you access buffer, or before the next two commands
                fmt▲      = DrawingBufferPixelFormat( )
              bytesInLine = DrawingBufferPitch( )
```

▲ PB_PixelFormat_(8Bits; palettized|15Bits|16Bits|24Bits_RGB|24its_BGR|32Bits_RGB|32Bits_BGR)

Image

; Max. image size: 8192 x 8192 [pix]
; Dimensions & coords are in pixels
; #PB_Image_DisplayFormat can result in color info loss
; For transparent GUI images, use *.ico's

; Images are used:
- for 2DDRAWING:DrawImage( ) on any graph. output
- for ImageGadget, ButtonImageGadget
- for item icons of ListIconGadget, TreeGadget
- for icons of ToolBar, StatusBar, SysTray
- with CLIPBOARD:Get-/SetClipboardImage( )
- for drawing on it (see IMAGE:ImageOutput( ))



create/free, load/save

; ! See also: CLIPBOARD:GetClipboardImage( )

```
                FreeImage    (#img)

imgID | 0 failed = CreateImage (#img, w, h, depth 1,2,4,8,16,24,32, or •#PB_Image_DisplayFormat ) ; Image is black
imgID | 0 failed = CatchImage  (#img, memAddr e.g.:?label1, length, flags▲) ; Often, used together w/ IncludeBinary

imgID | 0 failed = CopyImage   (#imgFrom, #imgTo)
imgID | 0 failed = GrabImage   (#imgFrom, #imgTo, x, y, w, h) ; With same depth

imgID | 0 failed = LoadImage▲(#img, fname$ *.bmp, *.ico, any IMAGEPLUGIN-supported, flags▲  ) ; • depth = 24 (32)
          ok? = SaveImage▲(#img, fname$, plugin▲, flags JPEG quality: 0(fastest)..10(best), •7) ; • depth = 24 (32)
```

▲ OR'ed flags (•0): #PB_Image_DisplayFormat
▲ #PB_ImagePlugin_(BMP;•,24 bits| JPEG| PNG)
▲ Use IMAGEPLUGIN lib's encoders/decoders for being able to use various graph. file formats

dimensions

```
w [pix]       = ImageDepth (#img)    ResizeImage(#img, w, h, mode #PB_Image_(•Smooth|Raw)) ; Changes image handle
h [pix]       = ImageDepth (#img)

bitsPerColor = ImageDepth (#img) ; 1,2,4,8,16,24,32
```

identification

```
is#imgValid?= IsImage      (#img)
outputID    = ImageOutput  (#img) ; For use in DRAWING:StartDrawing( ). Not for *.ico's!
imgID       = ImageID      (#img) ; For use in DRAWING:DrawImage( ), GADGET:ImageGadget( ), etc.
```

ImagePlugin

; ! Switch on necessary de-/encoders

decoders

; Decoders enhance the following commands:
IMAGE:Load-/CatchImage( ),
SPRITE:Load-/CatchSprite( )

```
UsePNGImageDecoder  ( ) ; PB ignores alphachannel
UseJPEGImageDecoder ( ) ; Lossy compression
UseTGAImageDecoder  ( ) ; PB ignores alphachannel
UseTIFFImageDecoder ( )
```

encoders

; Encoders enhance the following commands:
IMAGE:SaveImage( ),
SPRITE:SaveSprite( )

```
UsePNGImageEncoder  ( ) ; PB ignores alphachannel
UseJPEGImageEncoder ( ) ; Lossy compression
```

*. JPEG
*. PNG

*. TGA
*. TIFF

Movie

; DirectX 7+ is required (3+ on NT 4.0)
; Commands dealing with frames are invalid for frameless formats (like MP3)
; If you don't use video stream of a file, it's still possible to use its audio stream
; ! See also: AUDIOCD, SOUND (*.avi, plugin-supported), MODULE (tracker formats) libs

initialization

ok? = InitMovie( ) ; ! Call before other MOVIE commands

load/free

movieID | 0 file not found or format not supported = LoadMovie (#movie, fname$ *.avi, *.mpg, *.divx, *.mp3, etc.)
FreeMovie (#movie)

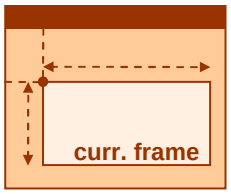
is#movieValid? = IsMovie(#movie)

control

PlayMovie (#movie, winID|screenID|#PB_Movie_Rendered ▲for RenderMovieFrame( ))
StopMovie (#movie)

PauseMovie (#movie)
ResumeMovie (#movie)

RenderMovieFrame ▲(#movie, spriteID) ; Cool effects with SPRITE3D commands are possible



static traits

len [frames] = MovieLength(#movie)

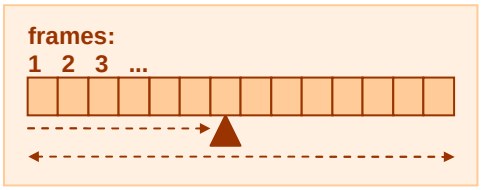
w [pix] | –1 no video stream = MovieWidth (#movie) ; Non-resized value
h [pix] | –1 no video stream = MovieHeight (#movie) ; Non-resized value

dynamic traits

ResizeMovie (#movie, x, y, w, h) ; In pix. on output surface
MovieAudio (#movie, volume 0..100, balance –100..0..100)

MovieSeek (#movie, frame 1..)
–1 paused | 0 stopped | frame current = MovieStatus (#movie)

FPS [1000*frames/sec] = MovieInfo(#movie, flags 0:ask for FPS)



AudioCD

; If >1 CD drives are installed, CDs can play simultaneously
; AUDIOCD commands apply to the current CD drive
; ! See also: MOVIE (*.avi, *.mpg, *.divx, *.mp3 ...), SOUND (*.avi, plugin-supported), MODULE (tracker formats) libs

initialization

numberOfAudioCDDrives▲| 0 no drives or failure = InitAudioCD( ) ; ! Call before other AUDIOCD commands

▲ Including virtual drives

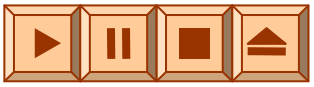
current drive selection

UseAudioCD(newCurrDriveIdx 0..) ; ! Don't rely on the default current drive, call it explicitly
drivePath$ e.g.: "D:\" = AudioCDName( )

control

PlayAudioCD▲ (startTrack 1.. , endTrack inclusive, 1..)
StopAudioCD ( )
PauseAudioCD ( )
ResumeAudioCD▲ ( )

EjectAudioCD (1 eject | 0 swallow disk)



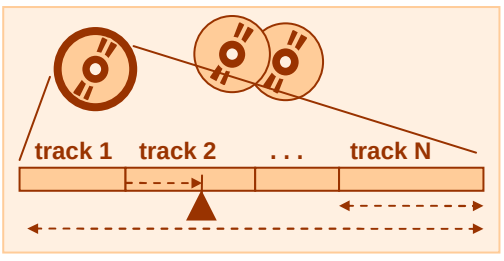
▲ Starts/resumes playing in the background and returns

static traits

numberOf = AudioCDTracks( )
trackLen [sec] = AudioCDTrackLength(trackIdx 1..)
fullLen [sec] = AudioCDLength( )

dynamic traits

–1 not ready | 0 stopped | track playing = AudioCDStatus( )
sinceCurrTrackStartedPlaying [sec] = AudioCDTrackSeconds( )



! For the rest of standard PB commands see the Full edition of MilliHelp

Legend

; This document represents infomation, heavily relying on colors.
  Thus, a color printer IS A MUST when making a paper copy! Otherwise, a part of the information will be lost.

metasymbols

; Optional parts are denoted by [ ]: A[B] means AB or A
; Alternatives are denoted by |: (A|B|C) means A or B or C; [A|B|C] means A or B or C or nothing

syntax coloring

; Each command name is of one of the four colors, according to the main (prevailing) function of the command:
  DoSomething( ), SetSomething( ), GetSomething( ), ConvertSomething( )

; Optional command parameters are in light color:
  SomeCommand(A, B, C, D) is same as SomeCommand(A, B[, C[, D]])
; Light comma between optional params means that they are either all present, or all omitted:
  SomeCommand(A, B, C, D) is same as SomeCommand(A, B[, C, D])

; In comments PB constants are in dark gray: #PB_SomeConst

comments

; Black comments begin with ";"
; Light gray comments are short and for commenting command parameters / return values just-in-place
; Footnote comments are marked by color triangles (▲▲▲▲) and located under a gray horizontal line.

Lib1

; Comment (applies to whole Lib1 topic)

▲ Footnote comment (applies to whole Lib1 topic)

SubTopic 1

; Comment (applies to SubTopic1)

```
z for z =  CommandA▲ (x comment for x, y▲)
           CommandB▲ (x, y) ; Comment (applies to CommandB( ))
           CommandC    (xx▲)
```

▲ Footnote comment (applies to SubTopic 1)
▲ #PB_SomeConst_(AA;comment for AA|BB|CC)

SubTopic 2

CommandD( )

some abbreviations

•	– by default
res	– result
ok?	– 0 if command fails, non-0 otherwise
yesNo	– 0 if yes, non-0 otherwise
isX?	– 0 if X, non-0 otherwise
w, h	– width, height
fname	– file name
dname	– directory name
0..	– one of the numbers: 0,1,2,3...
1..	– one of the numbers: 1,2,3,4...
X..Y	– any from X to Y
[pix]	– in pixels (Pixels are used as measure)

some icons

Library initialization

Create/Open/Load an object or resource

Free/Close an object or resource

ExamineXXX pattern

Event related stuff

drawings

; There are many drawings in this document. They serve as visual clues to the libraries.

other

; References to library commands have the following form: "LIBRARYNAME:CommandName( )"
; Notes that begin with "!" are in imperative mode: "! Do this and that"