Window

COLLABORATORS			
	TITLE : Window		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Window

1.1 Window

```
PureBasic Window library

  Windows are essentials component of modern interfaces. Pure Basic
  provide you a full access to them.

Commands summary:

   ActivateWindow
   BevelBox
   BusyPointer
   ChangeIDCMP
   CloseWindow
   DetachGadgetList
   DetachMenu
   EventCode
   EventGadgetID
   EventMenuID
   EventQualifier
   EventWindowID
   MoveWindow
   OpenWindow
   SetWindowTagList
   SizeWindow
   UseWindow
   WaitWindowEvent
   WindowBorderBottom
   WindowBorderLeft
   WindowBorderRight
   WindowBorderTop
   WindowEvent
   WindowHeight
   WindowID
   WindowInnerHeight
   WindowInnerWidth
   WindowMouseX
   WindowMouseY
   WindowRastPort
   WindowWidth
```

```
     WindowX
     WindowY

  Example:

     Window
```

1.2 bevelbox

```
    SYNTAX
  BevelBox(x, y, Width, Height, Style)

    COMMAND
  Display a standard BevelBox (ie: 3D like box, with highlight and shadow
  borders based on the system colours) on the current window.

  Two styles values are available:

    0: Raised (Normal 3D look)
    1: Recessed (3D Pushed look)
```

1.3 changeidcmp

```
    SYNTAX
  ChangeIDCMP(NewIDCMP)

    COMMAND
  Alter the PureBasic default IDCMP value for future opened window.
  For full description of IDCMPs click  here .
```

1.4 eventcode

```
    SYNTAX
  Code.l = EventCode()

    COMMAND
  After a WindowEvent() result, check this function to determine whether
  a gadtools gadget has been activated.

    ie: If the Gadget is:

    + CheckBox: EventCode() returns 1 or 0, to reflect the checkbox state
    + Palette : EventCode() returns the colour number that has been checked
    + Integer : EventCode() returns the number contained by the gadget
    + Option  : EventCode() returns the index of the selected option
    + ListView: EventCode() returns the index of the selected item
    + Slider  : EventCode() returns the current position of the slider
```

1.5 eventwindowid

```
  SYNTAX
#Window = EventWindowID()

  COMMAND
After a WindowEvent() result, check this function to determine on which
window the event has occur.
```

1.6 waitwindowevent

```
  SYNTAX
IDCMP.l = WaitWindowEvent()

  FUNCTION
Check if an event has occur on any of the open windows. To get
the window number in which the event occurred, you need to use
the EventWindowID() function.

This function is the same as WindowEvent() except that it will
wait, halt code execution, until an event occur.
```

1.7 windowborderbottom

```
  SYNTAX
Result.w = WindowBorderBottom()

  FUNCTION
Return the bottom border width in pixels, of the
currently used window.
```

1.8 windowborderleft

```
  SYNTAX
Result.w = WindowBorderLeft()

  FUNCTION
Return the left border width in pixels, of the
currently used window.
```

1.9 windowborderright

```
  SYNTAX
Result.w = WindowBorderRight()

  FUNCTION
Return the right border width in pixels, of the
currently used window.
```

1.10 windowbordertop

```
  SYNTAX
Result.w = WindowBorderTop()

  FUNCTION
Return the top border width in pixels, of the
currently used window.
```

1.11 windowevent

```
  SYNTAX
IDCMP.l = WindowEvent()

  FUNCTION
Check if an event has occur on any of the open windows. To get the window number
in which the event occurred, you need to use the EventWindowID()
function.

Most used IDCMP are: #IDCMP_GADGETUP (a gadget is pressed)
                     #IDCMP_CLOSEWINDOW (window’s closegadget is pushed)
                     #IDCMP_MENUPICK (a menu has be chosen)

For a full list and definition of IDCMP, look here:  IDCMP


Example:


InitWindow(0)
InitTagList(2)

ResetTagList(#WA_IDCMP, #IDCMP_CLOSEWINDOW | #IDCMP_MENUPICK | #IDCMP_GADGETUP)
If OpenWindow(0, 100, 100, 100, 100, #WFLG_CLOSEGADGET, TagListID())

  Repeat

    Repeat
      VWait()
      IDCMP.l = WindowEvent()
    Until IDCMP

  Until IDCMP = #IDCMP_CLOSEWINDOW

Endif

End
```

1.12 detachgadgetlist

```
  SYNTAX
DetachGadgetList()
```

```
  STATEMENT
Detach the current window's gadgetlist (if any). Note that the window
display isn't refreshed, so you can still see the gadgets, but you
can't push them any more (it's only the imagery which is visible).

You can use a combination of DetachGadgetList()/AttachGadgetList()
to change 'on the fly' a window gadgetlist.
```

1.13 openwindow

```
  SYNTAX
WindowID.l = OpenWindow(#Window, x, y, Width, Height, Flags, Title$)

  FUNCTION
Opens a new window according to the specified parameter. The new
window becomes the used window. You don't need to use the UseWindow()
command to set it. If the WindowID is NULL, the window can't be opened.

A TagList can be specified by using the SetWindowTagList() command
to have an extended control over the Window.
```

1.14 setwindowtaglist

```
  SYNTAX
SetWindowTagList(TagListID)

  STATEMENT
Pass a taglist which will be used by the next OpenWindow() call.

Available tags:

  #WA_Left
  #WA_Top
  #WA_Width
  #WA_Height
  #WA_DetailPen    - NOTE: only overrides NewWindow.DetailPen of -1!
  #WA_BlockPen     - NOTE: only overrides NewWindow.BlockPen of -1!
  #WA_IDCMP
  #WA_Flags        - initial values for Flags before looking at other
                     Boolean component Tag values
  #WA_Gadgets
  #WA_Title
  #WA_CustomScreen - also implies CUSTOMSCREEN property
  #WA_SuperBitMap  - also implies #WFLG_SUPER_BITMAP refresh mode.
  #WA_MinWidth
  #WA_MinHeight
  #WA_MaxWidth
  #WA_MaxHeight

  These Boolean tag items are alternatives to the NewWindow.Flags
  Boolean attributes with similar names.
```

```
#WA_SizeGadget       - equivalent to #WFLG_SIZEGADGET
#WA_DragBar          - equivalent to #WFLG_DRAGBAR
#WA_DepthGadget      - equivalent to #WFLG_DEPTHGADGET
#WA_CloseGadget      - equivalent to #WFLG_CLOSEGADGET
#WA_Backdrop         - equivalent to #WFLG_BACKDROP
#WA_ReportMouse      - equivalent to #WFLG_REPORTMOUSE
#WA_NoCareRefresh    - equivalent to #WFLG_NOCAREREFRESH
#WA_Borderless       - equivalent to #WFLG_BORDERLESS
#WA_Activate         - equivalent to #WFLG_ACTIVATE
#WA_RMBTrap          - equivalent to #WFLG_RMBTRAP
#WA_WBenchWindow     - equivalent to #WFLG_WBENCHWINDOW
              (system PRIVATE)
#WA_SimpleRefresh    - only specify if TRUE
#WA_SmartRefresh     - only specify if TRUE
#WA_SizeBRight       - equivalent to #WFLG_SIZEBRIGHT
#WA_SizeBBottom      - equivalent to #WFLG_SIZEBBOTTOM
#WA_GimmeZeroZero    - equivalent to #WFLG_GIMMEZEROZERO
#WA_NewLookMenus     - equivalent to #WFLG_NEWLOOKMENUS

The following tag items specify new attributes of a window.

#WA_ScreenTitle - You can specify the screen title associated
    with your window this way, and avoid a call to SetWindowTitles()
    when your window opens.

#WA_AutoAdjust - a Boolean attribute which says that it's OK
    to move or even shrink the dimensions of this window
    to fit it on the screen, within the dimension
    limits specified by MinWidth and MinHeight.
    Someday, this processing might be sensitive to the
    currently visible portion of the screen the window
    will be opening on, so don't draw too many conclusions
    about the auto-adjust algorithms.
    (Normally, this attribute defaults to FALSE.  However,
    if you call OpeWindowTags() or OpeWindowTagList()
    with a NULL NewWindow pointer, this attribute defaults
    to TRUE).

#WA_InnerWidth
#WA_InnerHeight - You can specify the dimensions of the interior
    region of your window, independent of what the border
    thicknesses will be.  You probably want to specify
    #WA_AutoAdjust to allow Intuition to move your window
    or even shrink it so that it is completely on screen.

    Note: using these tags puts some reasonable restrictions
    on the gadgets you can specify as "border" gadgets when
    you open your window.  Since border gadgets determine
    the border dimensions and hence the overall dimensions of
    your window, those dimensions cannot be used calculating
    the position or dimensions of border gadgets.

    Here's the complete list of restrictions:
    - #GACT_LEFTBORDER gadgets cannot be GFLG_RELWIDTH if #WA_InnerWidth is  ←
       used.
    - #GACT_RIGHTBORDER gadgets MUST be GFLG_RELRIGHT if #WA_InnerWidth is  ↩
       used.
    - #GACT_TOPBORDER gadgets cannot be GFLG_RELHEIGHT if #WA_InnerHeight is  ↩
       used.
    - #GACT_BOTTOMBORDER gadgets MUST be GFLG_RELBOTTOM if #WA_InnerHeight is  ↩
       used.

#WA_PubScreenName - This tag item declares that you want your window
    to open as a visitor window on the public screen whose name
    is pointed to by (UBYTE *) ti_Data.

#WA_PubScreen - Open as a visitor window on the public screen
    whose address if provided as (struct Screen *) ti_Data.
    To ensure that this screen remains open long enough, you
    must either:
    1) Be the screen's owner
    2) have another window already open on the screen
    3) use LockPubScreen()
    Using exec.library/Forbid() is not sufficient.

    You can provide ti_Data to be NULL (zero), without any
    of the above precautions, to specify the default public screen.

#WA_PubScreenFallBack - This Boolean attribute specifies that a
    visitor window should "fall back" to opening on the default
    public screen if the explicitly specify public screen is not
    available.

#WA_WindowName - this visionary specification of a window
    rendezvous name string is not yet implemented.

#WA_Colors - this equally great idea about associating a palette
    specification with the active window may not ever be implemented.

#WA_Zoom - ti_Data points to an array of four WORD's to be used
    as the initial Left/Top/Width/Height of the "alternate
    Zoom position and dimensions."  The presence of this tag
    item implies that you want a Zoom gadget, even though you
    might not have a sizing gadget.
    New for V39: if the initial zoom-box left and top are
    both set to ~0, then Intuition will give your window
    "size-only" zooming, meaning that zooming the window
    will not affect the left/top unless the window needs
    to be moved on-screen.

#WA_MouseQueue - This tag specifies a limit for the number
    of outstanding IDCMP_MOUSEMOVE IntuiMessages that Intuition
    will send to your window.  You can change the value of this
    limit after the window is open using SetMouseQueue().

#WA_RptQueue - This tag specifies a limit for the number of
    outstanding repeated-IDCMP_RAWKEY, repeated-IDCMP_VANILLAKEY,
    and repeated-IDCMP_IDCMPUPDATE IntuiMessages that Intuition will
    send to your window.  Currently, there is no function to adjust
    the repeat-key queue.

#WA_BackFill - ti_Data is a pointer to a Hook structure that
```

```
        the Layers library will call when your window needs
        "backfilling."  See layers.library/InstallLayerHook().

    #WA_MenuHelp - ti_Data is a boolean.  If true, enables the MenuHelp
        feature for this window.  See IDCMP_MENUHELP above.  (V37)

    #WA_NotifyDepth - ti_Data is a boolean.  Set to true if you
        would also like IDCMP_CHANGEWINDOW events sent to your window
        when it is depth-arranged.  Normally, such events are only
        sent for movement or resizing of the window.
        IDCMP_CHANGEWINDOW events originating from
        depth-arrangement have a Code equal to CWCODE_DEPTH, as
        opposed to CWCODE_MOVESIZE.  (V39)

    #WA_Checkmark - (ti_Data is struct Image *) Image to use as a
        checkmark in menus.  Prior to V39, or if #WA_NewLookMenus
        is not specified, the default will be the traditional
        checkmark in the original colors.  Under V39 and higher,
        if you have requested #WA_NewLookMenus then the default will
        be an appropriately colored checkmark scaled to the screen's
        font.  Alternately, you can provide a custom one, which you
        can~design yourself or get from sysiclass (use this if
        your menu-font is different from the screen's font).

    #WA_AmigaKey - (ti_Data is struct Image *) Image to use as
        the Amiga-key symbol in menus.  If #WA_NewLookMenus is not
        specified, the default will be the traditional Amiga-key
        symbol in the original colors.  If you've requested
        #WA_NewLookMenus, then the default will be an appropriately
        colored Amiga-key scaled to the screen's font.
        Alternately, you can provide a custom one, which you can
        design yourself or get from sysiclass (use this if your
        menu-font is different from the screen's font).  (V39)

    #WA_Pointer - (APTR) The pointer you wish to associate with
        your window.  If NULL, you are requesting the Preferences
        default pointer.  Custom pointers should be allocated by
        performing a NewObject() on "pointerclass".  (See
        <intuition/pointerclass.h>).  Defaults to NULL.  This tag is
        also recognized by SetWindowPointerA().  (V39)

    #WA_BusyPointer (BOOL) - Set to TRUE to request the Preferences
        busy-pointer.  If FALSE, your pointer will be as requested
        by #WA_Pointer.  Defaults to FALSE.  This tag is also
        recognized by SetWindowPointerA().  (V39)

    #WA_PointerDelay - (BOOL) Set to TRUE to defer changing your
        pointer for a brief instant.  This is typically used along
        with setting the busy pointer, especially when the
        application knows it may be busy for a very short while.  If
        the application clears the pointer or sets another pointer
        before the delay expires, the pending pointer change is
        cancelled.  This reduces short flashes of the busy pointer.
        This tag is also recognized by SetWindowPointerA().  (V39)

    #WA_HelpGroup - (ULONG) Normally, only the active window can
        receive IDCMP_GADGETHELP messages.  However, an application
        with multiple windows will want all its windows to be able
        to receive help when any of them are active.  First obtain a
        unique help ID with utility.library/GetUniqueID(), then
        pass it as ti_Data of this tag to all your windows.  See
        HelpControl().  (V39)

    #WA_HelpGroupWindow - (struct Window *) Instead of using
        #WA_HelpGroup, you can pass a pointer to another window
        whose HelpGroup you wish this window to belong to.  (V39)

    #WA_TabletMessages - (BOOL) Set to TRUE to request extended
        IntuiMessages for your window.  If a tablet driver is
        generating IESUBCLASS_NEWTABLET input events, you will be
        able to receive extended tablet information with most
        IntuiMessages.  See the eim_TabletData field of the
        ExtIntuiMessage structure.  Defaults to FALSE.  (V39)
```

1.15 windowmousex

```
    SYNTAX
  x.w = WindowMouseX()

    FUNCTION
  Returns the mouse position relative to the left of the current window.
  Values can be positive or negative.
```

1.16 windowmousey

```
    SYNTAX
  y.w = WindowMouseY()

    FUNCTION
  Returns the mouse position relative to the top of the current window.
  Values can be positive or negative.
```

1.17 windowwidth

```
    SYNTAX
  width.w = WindowWidth()

    FUNCTION
  Return the width, in pixels, of the current window.
```

1.18 windowheight

```
  SYNTAX
height.w = WindowHeight()

  FUNCTION
Returns the height, in pixels, of the current window.
```

1.19 windowx

```
  SYNTAX
x.w = WindowX()

  FUNCTION
Returns the left position, in pixels, of the current window.
```

1.20 windowy

```
  SYNTAX
y.w = WindowY()

  FUNCTION
Returns the top position, in pixels, of the current window.
```

1.21 windowrastport

```
  SYNTAX
rastport.l = WindowRastPort()

  FUNCTION
Returns the rastport of the current window.
```

1.22 movewindow

```
  SYNTAX
MoveWindow(x,y)

  STATEMENT
Move the window to the specified coordinates.
```

1.23 sizewindow

```
  SYNTAX
SizeWindow(Width, Height)

  STATEMENT
Resize the window to the given dimensions.
```

1.24 activatewindow

```
  SYNTAX
ActivateWindow()

  STATEMENT
Activate the current window.
```

1.25 usewindow

```
  SYNTAX
UseWindow(#Window)

  STATEMENT
Make the given window, the currently-used window.
```

1.26 closewindow

```
  SYNTAX
CloseWindow(#Window)

  STATEMENT
Close the given window.
```

1.27 windowid

```
  SYNTAX
WindowID.l = WindowID()

  FUNCTION
Returns the Intuition Window pointer.
```

1.28 busypointer

```
  SYNTAX
BusyPointer(State)

  FUNCTION
State = 0 or 1. If State = 1 a busypointer will be displayed for the
current window, otherwise the normal pointer will be displayed.
```

1.29 windowinnerheight

```
  SYNTAX
Result.w = WindowInnerHeight()

  FUNCTION
Returns the current window's inner-height, in pixels, (window's height
without the top and bottom border heights)
```

1.30 windowinnerwidth

```
  SYNTAX
Result.w = WindowInnerWidth()

  FUNCTION
Returns the used window's inner-width, in pixels, (window's width
without the left and right border widths)
```

1.31 detachmenu

```
  SYNTAX
DetachMenu()

  STATEMENT
Detach the menu from the currently used window. It's often used to
change the menu layout and re-attach the new menu (AttachMenu()).
```

1.32 eventqualifier

```
  SYNTAX
Qualifier = EventQualifier()

  STATEMENT
Get the 'Qualifier' from the last keypress. The Qualifier is the
alternate keys you press while you press a key (ie: shit, control..).
```

1.33 idcmp

```
IDCMP is an acronym for : 'Intuition Direct Communication Message Port'

Background: The Amiga interface system (called Intuition) communicates
            with the rest of the Amiga libraries through the message
            ports. All Windows opened have a message port which recieves
            all the information needed.
            For example, when you press the mouse button on a gadget,
            a message is sent to the window stating: 'A gadget has been
            pushed'. This is the role of the IDCMPs, which are constants,
            and each one corresponds to a different action. Here is
            the list of all the events that can happen in your window.

Note: to recieve the messages, you need to say it when you open your
window (with the #WA_IDCMP, <Your IDCMP you want to recieve here> inside
the taglist)

All these constants are in the AmigaLibs.res file.


IDCMP Description:

      - #IDCMP_NEWSIZE is the flag that tells Intuition to send an IDCMP
        message to you after the user has resized your window.  At
        this point, you could examine the size variables in your
        window structure to discover the new size of the window.
        See also the #IDCMP_CHANGEWINDOW IDCMP flag.

      - #IDCMP_REFRESHWINDOW when set will cause a message to be sent
        whenever your window needs refreshing.  This flag makes
        sense only with #WFLG_SIMPLE_REFRESH and #WFLG_SMART_REFRESH
        windows.

      - #IDCMP_MOUSEBUTTONS will get reports about mouse-button up/down
        events broadcast to you (Note:  only the ones that
        don't mean something to Intuition.  If the user clicks the
        select button over a gadget, Intuition deals with it and you
        don't find out about it through here).

      - #IDCMP_MOUSEMOVE will work only if you've set the
        #WFLG_REPORTMOUSE flag above, or if one of your gadgets has the
        #GACT_FOLLOWMOUSE flag set.  Then all mouse movements will be
        reported here, providing your window is active.

      - #IDCMP_GADGETDOWN means that when the User "selects" a gadget
        you've created with the #GACT_IMMEDIATE flag set, the fact
        will be broadcast through the IDCMP.

      - #IDCMP_GADGETUP means that when the user "releases" a gadget that
        you've created with the #GACT_RELVERIFY flag set, the fact
        will be broadcast through the IDCMP.  This message is
        only generated if the release is "good", such as releasing
        the select button over a Boolean gadget, or typing ENTER
        in a string gadget.

      - #IDCMP_MENUPICK selects that menu number data will be sent via
        the IDCMP.

      - #IDCMP_CLOSEWINDOW means broadcast the #IDCMP_CLOSEWINDOW event
        through the IDCMP rather than the console.

      - #IDCMP_RAWKEY selects that all #IDCMP_RAWKEY events are
        transmitted via the IDCMP.  Note that these are absolutely RAW
        keycodes, which you will have to translate before using.
        Setting this and the MOUSE flags effectively eliminates the need
        to open a Console device to get input from the keyboard and
        mouse.  Of course, in exchange you lose all of the console
        features, most notably the "cooking" of input data and
```

```
    the systematic output of text to your window.

  - #IDCMP_VANILLAKEY is for developers who don't want the hassle
    of #IDCMP_RAWKEYS.  This flag will return all the keycodes after
    translation via the current country-dependent keymap.  When
    you set this flag, you will get IntuiMessages where the Code
    field has a decoded ANSI character code representing the key
    struck on the keyboard.  Only codes that map to a single
    character are returned: you can't read such keys as HELP or
    the function keys with #IDCMP_VANILLAKEY.

    NEW FOR V36: If you have both #IDCMP_RAWKEY and #IDCMP_VANILLAKEY
    set, Intuition will send an #IDCMP_RAWKEY event for those
    *downstrokes* which do not map to single-byte characters
    ("non-vanilla" keys).  In this way you can easily detect cursor
    keys, function keys, and the Help key without sacrificing the
    convenience of #IDCMP_VANILLAKEY.  NB: A side-effect of having
    both #IDCMP_RAWKEY and #IDCMP_VANILLAKEY set is that you never
    hear #IDCMP_RAWKEY upstrokes, even for keys that caused
    #IDCMP_RAWKEY downstrokes.

  - #IDCMP_INTUITICKS gives you simple timer events from Intuition
    when your window is the active one; it may help you avoid
    opening and managing the timer device.  With this flag set,
    you will get only one queued-up INTUITICKS message at a
    time.  If Intuition notices that you've been sent an
    #IDCMP_INTUITICKS message and haven't replied to it, another
    message will not be sent.  Intuition receives timer events and
    considers sending you an #IDCMP_INTUITICKS message approximately
    ten times a second.

  - #IDCMP_DELTAMOVE gives raw (unscaled) input event delta X/Y
    values.  This is so you can detect mouse motion regardless of
    screen/window/display boundaries.  This works a little
    strangely: if you set both #IDCMP_MOUSEMOVE and #IDCMP_DELTAMOVE.
    IDCMPFlags, you will get #IDCMP_MOUSEMOVE messages with delta
    x/y values in the MouseX and MouseY fields of the
    IDCMPMessage.

  - #IDCMP_NEWPREFS indicates you wish to be notified when the
    system-wide Preferences changes.  For V36, there is a new
    environment mechanism to replace Preferences, which we
    recommend you consider using instead.

  - Set #IDCMP_ACTIVEWINDOW and #IDCMP_INACTIVEWINDOW to get messages
    when those events happen to your window.  Take care not to
    confuse this "ACTIVEWINDOW" with the familiar sounding, but
    totally different "WINDOWACTIVE" flag.  These two flags have
    been supplanted by "#IDCMP_ACTIVEWINDOW" and "#WFLG_WINDOWACTIVE".
    Use the new equivalent terms to avoid confusion.

  - Set #IDCMP_DISKINSERTED or #IDCMP_DISKREMOVED to learn when
    removable disks are inserted or removed, respectively.

  - #IDCMP_IDCMPUPDATE is a new class for V36 which is used as
    a channel of communication from custom and boopsi gadgets
    to your application.
```

```
- #IDCMP_CHANGEWINDOW is a new class for V36 that will be sent
  to your window whenever its dimensions or position are changed
  by the user or the functions SizeWindow(), MoveWindow(),
  ChangeWindowBox(), or ZipWindow().

- #IDCMP_MENUHELP is new for V37.  If you specify the #WA_MenuHelp
  tag when you open your window, then when the user presses the
  HELP key on the keyboard during a menu session, Intuition will
  terminate the menu session and issue this even in place of an
  #IDCMP_MENUPICK message.
- NEVER follow the NextSelect link for MENUHELP messages.
- You will be able to hear MENUHELP for ghosted menus.
  (This lets you tell the user why the option is ghosted.)
- Be aware that you can receive a MENUHELP message whose code
  corresponds to a menu header or an item that has sub-items
  (which does not happen for MENUPICK).  The code may also be
  MENUNULL.
- LIMITATION:  if the user extend-selects some checkmarked
  items with the mouse, then presses MENUHELP, your
  application will only hear the MENUHELP report.  You
  must re-examine the state of your checkmarks when you
  get a MENUHELP.
- Availability of MENUHELP in V36 is not directly
  controllable.  We apologize...

- #IDCMP_GADGETHELP is new for V39.  If you turn on
  gadget help for your window (using the HelpControl())
  function, then Intuition will send #IDCMP_GADGETHELP
  messages when the mouse passes over certain gadgets or
  your window.  The IntuiMessage->Code field is normally
  ~0, but a boopsi gadget can return any word value it wishes.

  Ordinarily, gadget help is only processed for the active
  window.  When Intuition has determined that the mouse is
  pointing at a gadget which has the GMORE_GADGETHELP
  property, you will be sent an #IDCMP_GADGETHELP message
  whose IAddress points to the gadget.  When the mouse is
  over your window but not over any help-aware gadget, you
  will be sent a message whose IAddress is the window
  itself.  When the mouse is not over your window,
  Intuition sends a message whose IAddress is zero.

  A multi-window application can use the #WA_HelpGroup or
  #WA_HelpGroupWindow tags to indicate that all its windows
  belong in a group.  (The help group identifier should be
  obtained with utility.library/GetUniqueID().) This makes
  Intuition test gadget help in all windows of the group
  when any one of them is the active one.  Inactive windows
  whose #WA_HelpGroup matches the active window's receive
  #IDCMP_GADGETHELP messages when the mouse is over that
  window or any of its help-aware gadgets.  The GADGETHELP
  message with an IAddress of zero means the mouse is not
  over the active window or any other window of the same
  group.  It is always sent to the active window (which is
  not necessarily the window in your group that last got a
  message).
  To maximize performance, gadget help is not checked
  while the mouse is travelling quickly, or if it has not
  moved at all since the last test.  As well, if Intuition
  discovers that the mouse is still over same gadget and
  that gadget does not wish to send a different
  IntuiMessage->Code from the last message, no new
  IntuiMessage is sent.

- #IDCMP_REQVERIFY is the flag which, like #IDCMP_SIZEVERIFY and ...

- #IDCMP_MENUVERIFY (see immediately below), specifies that you
  want to make sure that your graphical state is quiescent
  before something extraordinary happens.  In this case, the
  extraordinary event is that a rectangle of graphical data is
  about to be blasted into your Window.  If you're drawing
  directly into its screen, you probably will wish to make sure
  that you've ceased drawing before the user is allowed to bring
  up the DMRequest you've set up, and the same for when system
  has a request for the user.  Set this flag to ask for that
  verification step.

- #IDCMP_REQCLEAR is the flag you set to hear a message whenever
  a requester is cleared from your window.  If you are using
  #IDCMP_REQVERIFY to arbitrate access to your screen's bitmap, it
  is safe to start your output once you have heard an
  #IDCMP_REQCLEAR for each #IDCMP_REQSET.

- #IDCMP_REQSET is a flag that you set to receive a broadcast
  for each requester that is opened in your window.  Compare
  this with #IDCMP_REQCLEAR above.  This function is distinct
  from #IDCMP_REQVERIFY.  This functions merely tells you that a
  requester has opened, whereas #IDCMP_REQVERIFY requires you to
  respond before the requester is opened.

- #IDCMP_MENUVERIFY is the flag you set to have Intuition stop
  and wait for you to finish all graphical output to your
  window before rendering the menus.  Menus are currently
  rendered in the most memory-efficient way, which involves
  interrupting output to all windows in the screen before the
  menus are drawn.  If you need to finish your graphical
  output before this happens, you can set this flag to make
  sure that you do.

- #IDCMP_SIZEVERIFY means that you will be doing output to your
  window which depends on a knowledge of the current size of the
  window.  If the user wants to resize the window,  you may want
  to make sure that any queued output completes before the sizing
  takes place (critical text, for instance).  If this is the
  case, set this flag.   Then, when the user wants to size,
  Intuition will send you the #IDCMP_SIZEVERIFY message and Wait()
  until you reply that it's OK to proceed with the sizing. NOTE:
  when we say that Intuition will Wait() until you reply, what
  we're really saying is that user will WAIT until you reply, which
  suffers the great negative potential of User-Unfriendliness.
  So remember:  use this flag sparingly, and, as always with any
  IDCMP Message you receive, reply to it promptly!  Then, after
```

```
        user has sized the window, you can find out about it using
        #IDCMP_NEWSIZE.
```

1.34 eventmenuid

```
   SYNTAX
 Menu.l = EventMenuID()

   COMMAND
 After a WindowEvent() result, check this function to determine
 which menu that has been pushed.
```

1.35 eventgadgetid

```
   SYNTAX
 Gadget.l = EventGadgetID()

   COMMAND
 After a WindowEvent() result, check this function to
 determine which gadget that has been pushed.
```