

gadtools

COLLABORATORS

	<i>TITLE :</i> gadtools		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	gadtools	1
1.1	gadtools.doc	1
1.2	gadtools.library/CreateContext	1
1.3	gadtools.library/CreateGadgetA	2
1.4	gadtools.library/CreateMenusA	6
1.5	gadtools.library/DrawBevelBoxA	7
1.6	gadtools.library/FreeGadgets	8
1.7	gadtools.library/FreeMenus	9
1.8	gadtools.library/FreeVisualInfo	10
1.9	gadtools.library/GetVisualInfoA	10
1.10	gadtools.library/GT_BeginRefresh	11
1.11	gadtools.library/GT_EndRefresh	12
1.12	gadtools.library/GT_FilterIMsg	12
1.13	gadtools.library/GT_GetIMsg	13
1.14	gadtools.library/GT_PostFilterIMsg	14
1.15	gadtools.library/GT_RefreshWindow	15
1.16	gadtools.library/GT_ReplyIMsg	15
1.17	gadtools.library/GT_SetGadgetAttrsA	16
1.18	gadtools.library/LayoutMenuItemsA	18
1.19	gadtools.library/LayoutMenusA	19

Chapter 1

gadtools

1.1 gadtools.doc

CreateContext()	FreeVisualInfo()	GT_PostFilterIMsg()
CreateGadgetA()	GetVisualInfoA()	GT_RefreshWindow()
CreateMenusA()	GT_BeginRefresh()	GT_ReplyIMsg()
DrawBevelBoxA()	GT_EndRefresh()	GT_SetGadgetAttrsA()
FreeGadgets()	GT_FilterIMsg()	LayoutMenuItemsA()
FreeMenus()	GT_GetIMsg()	LayoutMenusA()

1.2 gadtools.library/CreateContext

NAME

CreateContext -- Create a place for GadTools context data. (V36)

SYNOPSIS

```
gad = CreateContext(glistpointer);  
D0                                A0
```

```
struct Gadget *CreateContext(struct Gadget **);
```

FUNCTION

Creates a place for GadTools to store any context data it might need for your window. In reality, an unselectable invisible gadget is created, with room for the context data. This function also establishes the linkage from a glist type pointer to the individual gadget pointers. Call this function before any of the other gadget creation calls.

INPUTS

glistptr - Address of a pointer to a Gadget, which was previously set to NULL. When all the gadget creation is done, you may use that pointer as your NewWindow.FirstGadget, or in intuition.library/AddGList(), intuition.library/RefreshGList(), FreeGadgets(), etc.

RESULT

gad - Pointer to context gadget, or NULL if failure.

EXAMPLE

```

struct Gadget *gad;
struct Gadget *glist = NULL;
gad = CreateContext(&glist);
/* Other creation calls go here */
if (gad)
{
    myNewWindow.FirstGadget = glist;
    if ( myWindow = OpenWindow(&myNewWindow) )
    {
        GT_RefreshWindow(win);
        /* other stuff */
        CloseWindow(myWindow);
    }
}
FreeGadgets(glist);

```

NOTES

BUGS

SEE ALSO

1.3 gadtools.library/CreateGadgetA

NAME

CreateGadgetA -- Allocate and initialize a gadtools gadget. (V36)
 CreateGadget -- Varargs stub for CreateGadgetA(). (V36)

SYNOPSIS

```

gad = CreateGadgetA(kind, previous, newgad, taglist)
D0                D0      A0      A1      A2

```

```

struct Gadget *CreateGadgetA(ULONG, struct Gadget *,
    struct NewGadget *, struct TagItem *);

```

```

gad = CreateGadget(kind, previous, newgad, firsttag, ...)

```

```

struct Gadget *CreateGadget(ULONG, struct Gadget *,
    struct NewGadget *, Tag, ...);

```

FUNCTION

CreateGadgetA() allocates and initializes a new gadget of the specified kind, and attaches it to the previous gadget. The gadget is created based on the supplied kind, NewGadget structure, and tags.

INPUTS

kind - to indicate what kind of gadget is to be created.
 previous - pointer to the previous gadget that this new gadget is to be attached to.
 newgad - a filled in NewGadget structure describing the desired gadget's size, position, label, etc.
 taglist - pointer to a TagItem list.

TAGS

All kinds:

GT_Underscore (GadTools V37 and higher only).

Indicates the symbol that precedes the character in the gadget label to be underscored. This would be to indicate keyboard equivalents for gadgets (note that GadTools does not process the keys - it just displays the underscore).

Example: To underscore the "M" in "Mode"...

```
ng.ng_GadgetText = "_Mode:";
gad = CreateGadget(..._KIND, &ng, prev,
    GT_Underscore, '_',
    ...
);
```

BUTTON_KIND (action buttons):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

CHECKBOX_KIND (on/off items):

GTCB_Checked (BOOL) - Initial state of checkbox, defaults to FALSE.

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

CYCLE_KIND (multiple state selections):

GTCY_Labels (STRPTR *) - Pointer to NULL-terminated array of strings that are the choices offered by the cycle gadget (required).

GTCY_Active (UWORD) - The ordinal number (counting from zero) of the initially active choice of a cycle gadget (defaults to zero).

GA_Disabled (BOOL) - (GadTools V37 and higher only)
Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

INTEGER_KIND (numeric entry):

GTIN_Number (ULONG) - The initial contents of the integer gadget (default zero).

GTIN_MaxChars (UWORD) - The maximum number of digits that the integer gadget is to hold (defaults to 10).

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

STRINGA_ExitHelp (BOOL) - (New for V37) Set to TRUE to have the help-key cause an exit from the integer gadget. You will then receive a GADGETUP with code = 0x5F (rawkey for help).

GA_TabCycle (BOOL) - (New for V37) Set to TRUE so that pressing <TAB> or <Shift-TAB> will activate the next or previous such gadget. (defaults to TRUE, unlike regular Intuition string gadgets, which default to FALSE).

LISTVIEW_KIND (scrolling list):

GTLV_Top (UWORD) - Top item visible in the listview (defaults to zero).

GTLV_Labels (struct List *) - List of labels whose ln_Name fields are to be displayed in the listview.

GTLV_ReadOnly (BOOL) - If TRUE, then listview is read-only.

GTLV_ScrollWidth (UWORD) - Width of scroll bar for listview.
Must be greater than zero (defaults to 16).

GTLV_ShowSelected (struct Gadget *) - NULL to have the currently selected item displayed beneath the listview, or pointer to

an already-created GadTools STRING_KIND gadget to have an editable display of the currently selected item.

GTLV_Selected (UWORD) - Ordinal number of currently selected item, or ~0 to have no current selection (defaults to ~0).

LAYOUTA_Spacing - Extra space to place between lines of listview (defaults to zero).

MX_KIND (mutually exclusive, radio buttons):

GTMX_Labels (STRPTR *) - Pointer to a NULL-terminated array of strings which are to be the labels beside each choice in a set of mutually exclusive gadgets.

GTMX_Active (UWORD) - The ordinal number (counting from zero) of the initially active choice of an mx gadget (Defaults to zero).

GTMX_Spacing (UWORD) - The amount of space between each choice of a set of mutually exclusive gadgets. This amount is added to the font height to produce the vertical shift between choices. (defaults to one).

LAYOUTA_Spacing - FOR COMPATIBILITY ONLY. Use GTMX_Spacing instead. The number of extra pixels to insert between each choice of a mutually exclusive gadget. This is added to the present gadget image height (9) to produce the true spacing between choices. (defaults to FontHeight-8, which is zero for 8-point font users).

NUMBER_KIND (read-only numeric):

GTNM_Number - A signed long integer to be displayed as a read-only number (default 0).

GTNM_Border (BOOL) - If TRUE, this flag asks for a recessed border to be placed around the gadget.

PALETTE_KIND (color selection):

GTPA_Depth (UWORD) - Number of bitplanes in the palette (defaults to 1).

GTPA_Color (UBYTE) - Initially selected color of the palette (defaults to 1).

GTPA_ColorOffset (UBYTE) - First color to use in palette (defaults to zero).

GTPA_IndicatorWidth (UWORD) - The desired width of the current-color indicator, if you want one to the left of the palette.

GTPA_IndicatorHeight (UWORD) - The desired height of the current-color indicator, if you want one above the palette.

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

SCROLLER_KIND (for scrolling through areas or lists):

GTSC_Top (WORD) - Top visible in area scroller represents (defaults to zero).

GTSC_Total (WORD) - Total in area scroller represents (defaults to zero).

GTSC_Visible (WORD) - Number visible in scroller (defaults to 2).

GTSC_Arrows (UWORD) - Asks for arrows to be attached to the scroller. The value supplied will be taken as the width of each arrow button for a horizontal scroller, or the height of each button for a vertical scroller (the other dimension will match the whole scroller).

PGA_Freedom - Whether scroller is horizontal or vertical. Choose LORIENT_VERT or LORIENT_HORIZ (defaults to horiz).

GA_Immediate (BOOL) - Hear every IDCMP_GADGETDOWN event from scroller (defaults to FALSE).

GA_RelVerify (BOOL) - Hear every IDCMP_GADGETUP event from scroller (defaults to FALSE).

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

SLIDER_KIND (to indicate level or intensity):

GTSL_Min (WORD) - Minimum level for slider (default 0).

GTSL_Max (WORD) - Maximum level for slider (default 15).

GTSL_Level (WORD) - Current level of slider (default 0).

GTSL_MaxLevelLen (UWORD) - Max. length in characters of level string when rendered beside slider.

GTSL_LevelFormat (STRPTR) - C-Style formatting string for slider level. Be sure to use the 'l' (long) modifier. This string is processed using exec/RawDoFmt(), so refer to that function for details.

GTSL_LevelPlace - One of PLACETEXT_LEFT, PLACETEXT_RIGHT, PLACETEXT_ABOVE, or PLACETEXT_BELOW, indicating where the level indicator is to go relative to slider (default to PLACETEXT_LEFT).

GTSL_DispFunc (LONG (*function)(struct Gadget *, WORD)) - Function to calculate level to be displayed. A number-of-colors slider might want to set the slider up to think depth, and have a (1 << n) function here. Defaults to none. Your function must take a pointer to gadget as the first parameter, the level (a WORD) as the second, and return the result as a LONG.

GA_Immediate (BOOL) - If you want to hear each slider IDCMP_GADGETDOWN event.

GA_RelVerify (BOOL) - If you want to hear each slider IDCMP_GADGETUP event.

PGA_Freedom - Set to LORIENT_VERT or LORIENT_HORIZ to have a vertical or horizontal slider.

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

STRING_KIND (text-entry):

GTST_String (STRPTR) - The initial contents of the string gadget, or NULL (default) if string is to start empty.

GTST_MaxChars (UWORD) - The maximum number of characters that the string gadget is to hold.

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).

STRINGA_ExitHelp (BOOL) - (New for V37) Set to TRUE to have the help-key cause an exit from the string gadget. You will then receive a GADGETUP with code = 0x5F (rawkey for help).

GA_TabCycle (BOOL) - (New for V37) Set to TRUE so that pressing <TAB> or <Shift-TAB> will activate the next or previous such gadget. (defaults to TRUE, unlike regular Intuition string gadgets, which default to FALSE).

TEXT_KIND (read-only text):

GTTX_Text - Pointer to a NULL terminated string to be displayed, as a read-only text-display gadget, or NULL. defaults to NULL.

GTTX_CopyText (BOOL) - This flag instructs the text-display gadget to copy the supplied text string, instead of using only pointer to the string. This only works for the initial value of GTTX_Text set at CreateGadget() time. If you subsequently

change `GTTX_Text`, the new text will be referenced by pointer, not copied. Do not use this tag with a `NULL GTTX_Text`.
`GTTX_Border` (BOOL) - If `TRUE`, this flag asks for a recessed border to be placed around the gadget.

RESULT

`gad` - pointer to the new gadget, or `NULL` if the allocation failed or if previous was `NULL`.

EXAMPLE

NOTES

Note that the `ng_VisualInfo` and `ng_TextAttr` fields of the `NewGadget` structure must be set to valid `VisualInfo` and `TextAttr` pointers, or this function will fail.

Starting with V37, string and integer gadgets have the `GFLG_TABCYCLE` feature automatically. If the user presses `Tab` or `Shift-Tab` while in a string or integer gadget, the next or previous one in sequence will be activated. You will hear a `GADGETUP` with a code of `0x09`. Use `{GA_TabCycle, FALSE}` to suppress this.

BUGS

SEE ALSO

`FreeGadgets()`, `GT_SetGadgetAttrs()`, `GetVisualInfo()`.

1.4 gadtools.library/CreateMenuA

NAME

`CreateMenuA` -- Allocate and fill out a menu structure. (V36)
`CreateMenus` -- Varargs stub for `CreateMenus()`. (V36)

SYNOPSIS

```
menu = CreateMenuA(newmenu, taglist)
D0          A0          A1

struct Menu *CreateMenuA(struct NewMenu *, struct TagItem *);

menu = CreateMenus(newmenu, firsttag, ...)

struct Menu *CreateMenus(struct NewMenu *, Tag, ...);
```

FUNCTION

`CreateMenuA()` allocates and initializes a complete menu structure based on the supplied array of `NewMenu` structures. Optionally, `CreateMenuA()` can allocate and initialize a complete set of menu items and sub-items for a single menu title. This is dictated by the contents of the array of `NewMenus`.

INPUTS

`newmenu` - Pointer to an array of initialized `struct NewMenus`.
`taglist` - Pointer to a `TagItem` list.

TAGS

GTMN_FrontPen (UBYTE) - Pen number to be used for menu text.
(defaults to zero).

GTMN_FullMenu (BOOL) - (GadTools V37 and higher only)
Requires that the NewMenu specification describes a complete menu strip, not a fragment. If a fragment is found, CreateMenusA() will fail with a secondary error of GTMENU_INVALID. (defaults to FALSE).

GTMN_SecondaryError (ULONG *) - (GadTools V37 and higher only)
Supply a pointer to a NULL-initialized ULONG to receive a descriptive error code. Possible values:
GTMENU_INVALID - NewMenu structure describes an illegal menu. (CreateMenusA() will fail with a NULL result).
GTMENU_TRIMMED - NewMenu structure has too many menus, items, or subitems (CreateMenusA() will succeed, returning a trimmed-down menu structure).
GTMENU_NOMEM - CreateMenusA() ran out of memory.

RESULT

menu - Pointer to the resulting initialized menu structure (or the resulting FirstItem), with all the links for menu items and subitems in place.
The result will be NULL if CreateMenusA() could not allocate memory for the menus, or if the NewMenu array had an illegal arrangement (eg. NM_SUB following NM_TITLE).
(see also the GTMN_SecondaryError tag above).

EXAMPLE

NOTES

The strings you supply for menu text are not copied, and must be preserved for the life of the menu.
The resulting menus have no positional information. You will want to call LayoutMenusA() (or LayoutMenuItemsA()) to supply that. CreateMenusA() automatically provides you with a UserData field for each menu, menu-item or sub-item. Use the GTMENU_USERDATA(menu) or GTMENUITEM_USERDATA(menuitem) macro to access it.

BUGS

At present, if you put images into menus using IM_ITEM or IM_SUB for a NewMenu->Type, the image you supply must be an ordinary struct Image. You may not use a 'custom image' (eg. one obtained from a boopsi image-class).

SEE ALSO

LayoutMenusA(), FreeMenus(), gadtools.h/GTMENU_USERDATA(),
gadtools.h/GTMENUITEM_USERDATA()

1.5 gadtools.library/DrawBevelBoxA

NAME

DrawBevelBoxA -- Draws a bevelled box. (V36)
DrawBevelBox -- Varargs stub for DrawBevelBox(). (V36)

SYNOPSIS

DrawBevelBoxA(rport, left, top, width, height, taglist)

A0 D0 D1 D2 D3 A1

```
VOID DrawBevelBoxA(struct RastPort *, WORD, WORD, WORD, WORD,
    struct TagItem *taglist);
```

```
DrawBevelBox(rport, left, top, width, height, firsttag, ...)
```

```
VOID DrawBevelBox(struct RastPort *, WORD, WORD, WORD, WORD,
    Tag, ...);
```

FUNCTION

DrawBevelBoxA() renders a bevelled box of specified dimensions into the supplied RastPort.

INPUTS

rport - The RastPort into which the box is to be drawn.
 left - The left edge of the box.
 top - The top edge of the box.
 width - The width of the box.
 height - The height of the box.
 taglist - Pointer to a TagItem list.

TAGS

GTBB_Recessed (BOOL): Set to anything for a recessed-looking box. If absent, the box defaults, it would be raised.
 GT_VisualInfo (APTR): You MUST supply the value you obtained from an earlier call to GetVisualInfoA().

RESULT

None.

EXAMPLE

NOTES

DrawBevelBox() is a rendering operation, not a gadget. That means you must refresh it at the appropriate time, like any other rendering operation.

BUGS

SEE ALSO

GetVisualInfoA()

1.6 gadtools.library/FreeGadgets

NAME

FreeGadgets -- Free a linked list of gadgets. (V36)

SYNOPSIS

```
FreeGadgets(glist)
    A0
```

```
VOID FreeGadgets(struct Gadget *glist);
    A0
```

FUNCTION

Frees any GadTools gadgets found on the linked list of gadgets beginning with the specified one. Frees all the memory that was allocated by CreateGadgetA(). This function will return safely with no action if it receives a NULL parameter.

INPUTS

glist - pointer to first gadget in list to be freed.

RESULT

none

EXAMPLE

NOTES

BUGS

SEE ALSO

CreateGadgetA()

1.7 gadtools.library/FreeMenus

NAME

FreeMenus -- Frees memory allocated by CreateMenusA(). (V36)

SYNOPSIS

FreeMenus(menu)
A0

VOID FreeMenus(struct Menu *);

FUNCTION

Frees the menus allocated by CreateMenusA(). It is safe to call this function with a NULL parameter.

INPUTS

menu - Pointer to menu structure (or first MenuItem) obtained from CreateMenusA().

RESULT

None.

EXAMPLE

NOTES

BUGS

SEE ALSO

CreateMenusA()

1.8 gadtools.library/FreeVisualInfo

NAME

FreeVisualInfo -- Return any resources taken by GetVisualInfo. (V36)

SYNOPSIS

```
FreeVisualInfo(vi)
                A0
```

```
VOID FreeVisualInfo(APTR);
```

FUNCTION

FreeVisualInfo() returns any memory or other resources that were allocated by GetVisualInfoA(). You should only call this function once you are done with using the gadgets (i.e. after CloseWindow()), but while the screen is still valid (i.e. before CloseScreen() or UnlockPubScreen()).

INPUTS

vi - Pointer that was obtained by calling GetVisualInfoA().

RESULT

None.

EXAMPLE

NOTES

BUGS

SEE ALSO

GetVisualInfoA()

1.9 gadtools.library/GetVisualInfoA

NAME

GetVisualInfoA -- Get information GadTools needs for visuals. (V36)
GetVisualInfo -- Varargs stub for GetVisualInfoA(). (V36)

SYNOPSIS

```
vi = GetVisualInfoA(screen, taglist)
D0                A0        A1
```

```
APTR vi = GetVisualInfoA(struct Screen *, struct TagItem *);
```

```
vi = GetVisualInfo(screen, firsttag, ...)
```

```
APTR vi = GetVisualInfo(struct Screen *, Tag, ...);
```

FUNCTION

Get a pointer to a (private) block of data containing various bits of information that GadTools needs to ensure the best quality visuals. Use the result in the NewGadget structure of any gadget you create, or as a parameter to the various menu calls. Once the

gadgets/menus are no longer needed (after the last CloseWindow), call FreeVisualInfo().

INPUTS

screen - Pointer to the screen you will be opening on.
taglist - Pointer to list of TagItems.

RESULT

vi - Pointer to private data.

EXAMPLE

NOTES

BUGS

SEE ALSO

FreeVisualInfo(), intuition/LockPubScreen(),
intuition/UnlockPubScreen()

1.10 gadtools.library/GT_BeginRefresh

NAME

GT_BeginRefresh -- Begin refreshing friendly to GadTools. (V36)

SYNOPSIS

```
GT_BeginRefresh(win)
                A0
```

```
VOID GT_BeginRefresh(struct Window *);
```

FUNCTION

Invokes the intuition.library/BeginRefresh() function in a manner friendly to the Gadget Toolkit. This function call permits the GadTools gadgets to refresh themselves at the correct time. Call GT_EndRefresh() function when done.

INPUTS

win - Pointer to Window structure for which a IDCMP_REFRESHWINDOW IDCMP event was received.

RESULT

None.

EXAMPLE

NOTES

The nature of GadTools precludes the use of the IDCMP flag WFLG_NOCAREREFRESH. You must handle IDCMP_REFRESHWINDOW events in at least the minimal way, namely:

```
case IDCMP_REFRESHWINDOW:
    GT_BeginRefresh(win);
    GT_EndRefresh(win, TRUE);
    break;
```

BUGS

SEE ALSO

`intuition.library/BeginRefresh()`

1.11 gadtools.library/GT_EndRefresh

NAME

`GT_EndRefresh` -- End refreshing friendly to GadTools. (V36)

SYNOPSIS

```
GT_EndRefresh(win, complete)
               A0    D0
```

```
VOID GT_EndRefresh(struct Window *, BOOL complete);
```

FUNCTION

Invokes the `intuition.library/EndRefresh()` function in a manner friendly to the Gadget Toolkit. This function call permits GadTools gadgets to refresh themselves at the correct time. Call this function to `EndRefresh()` when you have used `GT_BeginRefresh()`.

INPUTS

`win` - Pointer to Window structure for which a `IDCMP_REFRESHWINDOW` `IDCMP` event was received.
`complete` - `TRUE` when done with refreshing.

RESULT

None.

EXAMPLE

NOTES

BUGS

SEE ALSO

`intuition.library/EndRefresh()`

1.12 gadtools.library/GT_FilterIMsg

NAME

`GT_FilterIMsg` -- Filter an `IntuiMessage` through GadTools. (V36)

SYNOPSIS

```
modimsg = GT_FilterIMsg(imsg)
D0                      A1
```

```
struct IntuiMessage *GT_FilterIMsg(struct IntuiMessage *);
```

FUNCTION

NOTE WELL: Extremely few programs will actually need this function. You almost certainly should be using `GT_GetIMsg()` and `GT_ReplyIMsg()` only, and not `GT_FilterIMsg()` and `GT_PostFilterIMsg()`.

`GT_FilterIMsg()` takes the supplied `IntuiMessage` and asks the Gadget Toolkit to consider and possibly act on it. Returns `NULL` if the message was only of significance to a GadTools gadget (i.e. not to you), else returns a pointer to a modified IDCMP message, which may contain additional information. You should examine the `Class`, `Code`, and `IAddress` fields of the returned message to learn what happened. Do not make interpretations based on the original `imsg`. You should use `GT_PostFilterIMsg()` to revert to the original `IntuiMessage` once you are done with the modified one.

INPUTS

`imsg` - An `IntuiMessage` you obtained from a Window's UserPort.

RESULT

`modimsg` - A modified `IntuiMessage`, possibly with extra information from GadTools, or `NULL`.

EXAMPLE

NOTES

BUGS

SEE ALSO

`GT_GetIMsg()`, `GT_PostFilterIMsg()`

1.13 gadtools.library/GT_GetIMsg

NAME

`GT_GetIMsg` -- Get an `IntuiMessage`, with GadTools processing. (V36)

SYNOPSIS

```
imsg = GT_GetIMsg(intuiport)
D0          A0
```

```
struct IntuiMessage *GT_GetIMsg(struct MsgPort *);
```

FUNCTION

Use `GT_GetIMsg()` in place of the usual `exec.library/GetMsg()` when reading `IntuiMessages` from your window's UserPort. If needed, the GadTools dispatcher will be invoked, and suitable processing will be done for gadget actions. This function returns a pointer to a modified `IntuiMessage` (which is a copy of the original, possibly with some supplementary information from GadTools). If there are no messages (or if the only messages are meaningful only to GadTools, `NULL` will be returned.

INPUTS

`intuiport` - The Window->UserPort of a window that is using the

Gadget Toolkit.

RESULT

imsg - Pointer to modified IntuiMessage, or NULL if there are no applicable messages.

EXAMPLE

NOTES

Be sure to use GT_ReplyIMsg() and not exec.library/ReplyMsg() on messages obtained with GT_GetIMsg().
If you intend to do more with the resulting message than read its fields, act on it, and reply it, you may find GT_FilterIMsg() more appropriate.

BUGS

SEE ALSO

GT_ReplyIMsg(), GT_FilterIMsg()

1.14 gadtools.library/GT_PostFilterIMsg

NAME

GT_PostFilterIMsg -- Return the unfiltered message after GT_FilterIMsg() was called, and clean up. (V36)

SYNOPSIS

```
imsg = GT_PostFilterIMsg(modimsg)
D0                                     A1
```

```
struct IntuiMessage *GT_PostFilterIMsg(struct IntuiMessage *);
```

FUNCTION

NOTE WELL: Extremely few programs will actually need this function. You almost certainly should be using GT_GetIMsg() and GT_ReplyIMsg() only, and not GT_FilterIMsg() and GT_PostFilterIMsg().

Performs any clean-up necessitated by a previous call to GT_FilterIMsg(). The original IntuiMessage is now yours to handle. Do not interpret the fields of the original IntuiMessage, but rather use only the one you got from GT_FilterIMsg(). You may only do message related things at this point, such as queueing it up or replying it. Since you got the message with exec.library/GetMsg(), your responsibilities do include replying it with exec.library/ReplyMsg(). This function may be safely called with a NULL parameter.

INPUTS

modimsg - A modified IntuiMessage obtained with GT_FilterIMsg().

RESULT

imsg - A pointer to the original IntuiMessage, if GT_FilterIMsg() returned non-NULL.

EXAMPLE

NOTES

Be sure to use `exec.library/ReplyMsg()` on the original `IntuiMessage` you obtained with `GetMsg()`, (which is the what you passed to `GT_FilterIMsg()`), and not on the parameter of this function.

BUGS

SEE ALSO

`GT_FilterIMsg()`

1.15 gadtools.library/GT_RefreshWindow

NAME

`GT_RefreshWindow` -- Refresh all the GadTools gadgets. (V36)

SYNOPSIS

```
GT_RefreshWindow(win, req)
                A0    A1
```

```
VOID GT_RefreshWindow(struct Window *, struct Requester *);
```

FUNCTION

Perform the initial refresh of all the GadTools gadgets you have created. After you have opened your window, you must call this function. Or, if you have opened your window without gadgets, you add the gadgets with `intuition.library/AddGLList()`, refresh them using `intuition.library/RefreshGLList()`, then call this function.
You should not need this function at other times.

INPUTS

`win` - Pointer to the Window containing GadTools gadgets.
`req` - Pointer to requester, or NULL if not a requester (currently ignored - use NULL).

RESULT

None.

EXAMPLE

NOTES

`req` must currently be NULL. GadTools gadgets are not supported in requesters. This field may allow such support at a future date.

BUGS

SEE ALSO

`GT_BeginRefresh()`

1.16 gadtools.library/GT_ReplyIMsg

NAME

GT_ReplyIMsg -- Reply a message obtained with GT_GetIMsg(). (V36)

SYNOPSIS

```
GT_ReplyIMsg(imsg)
            A1
```

```
VOID GT_ReplyIMsg(struct IntuiMessage *);
```

FUNCTION

Reply a modified IntuiMessage obtained with GT_GetIMsg().
If you use GT_GetIMsg(), use this function where you would normally have used exec.library/ReplyMsg().
You may safely call this routine with a NULL pointer (nothing will be done).

INPUTS

imsg - A modified IntuiMessage obtained with GT_GetIMsg().

RESULT

None.

EXAMPLE

NOTES

When using GadTools, you MUST explicitly GT_ReplyIMsg() all messages you receive. You cannot depend on CloseWindow() to handle messages you have not replied.

BUGS

SEE ALSO

GT_GetIMsg()

1.17 gadtools.library/GT_SetGadgetAttrsA

NAME

GT_SetGadgetAttrsA -- Change the attributes of a GadTools gadget. (V36)
GT_SetGadgetAttrs -- Varargs stub for GT_SetGadgetAttrsA(). (V36)

SYNOPSIS

```
GT_SetGadgetAttrsA(gad, win, req, taglist)
                A0    A1    A2    A3
```

```
VOID GT_SetGadgetAttrsA(struct Gadget *, struct Window *,
    struct Requester *, struct TagItem *);
```

```
GT_SetGadgetAttrs(gad, win, req, firsttag, ...)
```

```
VOID GT_SetGadgetAttrs(struct Gadget *, struct Window *,
    struct Requester *, Tag, ...);
```

FUNCTION

Change the attributes of the specified gadget, according to the

attributes chosen in the tag list.

INPUTS

gad - Pointer to the gadget in question.
win - Pointer to the window containing the gadget.
req - Pointer to the requester containing the gadget, or NULL if
not in a requester. (Not yet implemented, use NULL).
taglist - Pointer to TagItem list.

TAGS

BUTTON_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

CHECKBOX_KIND:

GTCB_Checked (BOOL) - Initial state of checkbox, defaults to FALSE.
GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

CYCLE_KIND:

GTCY_Active (UWORD) - The ordinal number (counting from zero) of
the active choice of a cycle gadget (defaults to zero).
GTCY_Labels (STRPTR *) - (GadTools V37 and higher only)
Pointer to NULL-terminated array of strings
that are the choices offered by the cycle gadget.
GA_Disabled (BOOL) - (GadTools V37 and higher only)
Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

INTEGER_KIND:

GTIN_Number (ULONG) - The initial contents of the integer gadget
(default zero).
GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

LISTVIEW_KIND:

GTLV_Top (UWORD) - Top item visible in the listview (defaults to zero).
GTLV_Labels (struct List *) - List of labels whose ln_Name fields
are to be displayed in the listview. Use a value of ~0 to
"detach" your List from the display. You must detach your list
before modifying the List structure, since GadTools reserves
the right to traverse it on another task's schedule. When you
are done, attach the list by using the tag pair
{GTLV_Labels, list}.
GTLV_Selected (UWORD) - Ordinal number of currently selected
item (defaults to zero if GTLV_ShowSelected is set).

MX_KIND:

GTMX_Active (UWORD) - The ordinal number (counting from zero) of
the active choice of an mx gadget (Defaults to zero).

NUMBER_KIND:

GTNM_Number - A signed long integer to be displayed as a read-only
number (default 0).

PALETTE_KIND:

GTPA_Color (UBYTE) - Initially selected color of the palette

(defaults to 1).

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

SCROLLER_KIND:

GTSC_Top (WORD) - Top visible in scroller (defaults to zero).

GTSC_Total (WORD) - Total in scroller area (defaults to zero).

GTSC_Visible (WORD) - Number visible in scroller (defaults to 2).

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

SLIDER_KIND:

GTSL_Min (WORD) - Minimum level for slider (default 0).

GTSL_Max (WORD) - Maximum level for slider (default 15).

GTSL_Level (WORD) - Current level of slider (default 0).

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

STRING_KIND:

GTST_String (STRPTR) - The initial contents of the string gadget,
or NULL (default) if string is to start empty.

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise
(defaults to FALSE).

TEXT_KIND:

GTTX_Text - Pointer to a NULL terminated string to be displayed,
as a read-only text-display gadget, or NULL. defaults to NULL.

RESULT

None.

EXAMPLE

NOTES

req must currently be NULL. GadTools gadgets are not supported
in requesters. This field may allow such support at a future date.

This function may not be called inside of a GT_BeginRefresh() /
GT_EndRefresh() session. (As always, restrict yourself to simple
rendering functions).

BUGS

SEE ALSO

1.18 gadtools.library/LayoutMenuItemsA

NAME

LayoutMenuItemsA -- Position all the menu items. (V36)

LayoutMenuItems -- Varargs stub for LayoutMenuItemsA(). (V36)

SYNOPSIS

```
success = LayoutMenuItemsA(menuitem, vi, taglist)
D0                      A0          A1  A2
```

```
BOOL LayoutMenuItemsA(struct MenuItem *, APTR, struct TagItem *);
```

```
success = LayoutMenuItems(menuitem, vi, firsttag, ...)
```

```
BOOL LayoutMenuItemsA(struct MenuItem *, APTR, Tag, ...);
```

FUNCTION

Lays out all the menu items and sub-items according to the supplied visual information and tag parameters. You would use this if you used CreateMenusA() to make a single menu-pane (with sub-items, if any), instead of a whole menu strip.

This routine attempts to columnize and/or shift the MenuItems in the event that a menu would be too tall or too wide.

INPUTS

menuitem - Pointer to first MenuItem in a linked list of items.

vi - Pointer returned by GetVisualInfoA().

taglist - Pointer to a TagItem list.

TAGS

GTMN_TextAttr (struct TextAttr *) - Text Attribute to use for menu-items and sub-items. If not supplied, the screen's font will be used. This font must be openable via OpenFont() when this function is called.

GTMN_Menu (struct Menu *) - Pointer to the Menu structure whose FirstItem is the MenuItem supplied above. If the menu items are such that they need to be columnized or shifted, the Menu structure is needed to perform the complete calculation. It is suggested you always provide this information.

RESULT

success - TRUE if successful, false otherwise (signifies that the TextAttr wasn't openable).

EXAMPLE

NOTES

BUGS

If a menu ends up being wider than the whole screen, it will run off the right-hand side.

SEE ALSO

CreateMenusA(), GetVisualInfoA()

1.19 gadtools.library/LayoutMenuA

NAME

LayoutMenuA -- Position all the menus and menu items. (V36)

LayoutMenu -- Varargs stub for LayoutMenuA(). (V36)

SYNOPSIS

```
success = LayoutMenuA(menu, vi, taglist)
```

```
D0                      A0      A1  A2
```

```
BOOL LayoutMenusA(struct Menu *, APTR, struct TagItem *);
```

```
success = LayoutMenus(menu, vi, firsttag, ...)
```

```
BOOL LayoutMenus(struct Menu *, APTR, Tag, ...);
```

FUNCTION

Lays out all the menus, menu items and sub-items in the supplied menu according to the supplied visual information and tag parameters. This routine attempts to columnize and/or shift the MenuItems in the event that a menu would be too tall or too wide.

INPUTS

menu - Pointer to menu obtained from CreateMenusA().
vi - Pointer returned by GetVisualInfoA().
taglist - Pointer to a TagItem list.

TAGS

GTMMN_TextAttr (struct TextAttr *) - Text Attribute to use for menu-items and sub-items. If not supplied, the screen's font will be used. This font must be openable via OpenFont() when this function is called.

RESULT

success - TRUE if successful, false otherwise (signifies that the TextAttr wasn't openable).

EXAMPLE

NOTES

When using this function, there is no need to also call LayoutMenuItemsA().

BUGS

If a menu ends up being wider than the whole screen, it will run off the right-hand side.

SEE ALSO

CreateMenusA(), GetVisualInfoA()