

gadtools

COLLABORATORS

	<i>TITLE :</i> gadtools		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 19, 2024	

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	8
1.5	gadtools.library/DrawBevelBoxA	10
1.6	gadtools.library/FreeGadgets	11
1.7	gadtools.library/FreeMenus	11
1.8	gadtools.library/FreeVisualInfo	12
1.9	gadtools.library/GetVisualInfoA	12
1.10	gadtools.library/GT_BeginRefresh	13
1.11	gadtools.library/GT_EndRefresh	14
1.12	gadtools.library/GT_FilterIMsg	14
1.13	gadtools.library/GT_GetGadgetAttrsA	15
1.14	gadtools.library/GT_GetIMsg	17
1.15	gadtools.library/GT_PostFilterIMsg	18
1.16	gadtools.library/GT_RefreshWindow	19
1.17	gadtools.library/GT_ReplyIMsg	20
1.18	gadtools.library/GT_SetGadgetAttrsA	20
1.19	gadtools.library/LayoutMenuItemsA	23
1.20	gadtools.library/LayoutMenusA	24

Chapter 1

gadtools

1.1 gadtools.doc

```
CreateContext ()
CreateGadgetA ()
CreateMenuSA ()
DrawBevelBoxA ()
FreeGadgets ()
FreeMenus ()
FreeVisualInfo ()
GetVisualInfoA ()
GT_BeginRefresh ()
GT_EndRefresh ()
GT_FilterIMsg ()
GT_GetGadgetAttrsA ()
GT_GetIMsg ()
GT_PostFilterIMsg ()
GT_RefreshWindow ()
GT_ReplyIMsg ()
GT_SetGadgetAttrsA ()
LayoutMenuItemsA ()
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, NULL);
        /* other stuff */
        CloseWindow(myWindow);
    }
}
FreeGadgets(glist);
```

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 - kind of gadget is to be created, one of the XXX_KIND values defined in <libraries/gadtools.h>
 previous - pointer to the previous gadget that this new gadget is to be attached to. This function will fail if this value is NULL
 newgad - a filled in NewGadget structure describing the desired gadget's size, position, label, etc.
 tagList - pointer to an array of tags providing optional extra parameters, or NULL

TAGS

All kinds:

GT_Underscore - Indicates the symbol that precedes the character in the gadget label to be underscored. This can be to indicate keyboard equivalents for gadgets (note that GadTools does not process the keys - it just displays the underscore). For example, to underscore the "M" in "Mode":

```
ng.ng_GadgetText = "_Mode:";
gad = CreateGadget(..._KIND, &ng, prev,
  GT_Underscore, '_',
  ...
);
(V37)
```

BUTTON_KIND (action buttons):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V36)
 GA_Immediate (BOOL) - Hear IDCMP_GADGETDOWN events from button gadget (defaults to FALSE). (V39)

CHECKBOX_KIND (on/off items):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE).
 GTCB_Checked (BOOL) - Initial state of checkbox (defaults to FALSE) (V36)
 GTCB_Scaled (BOOL) - If true, then checkbox imagery will be scaled to fit the gadget's width & height. Otherwise, a fixed size of CHECKBOXWIDTH by CHECKBOXHEIGHT will be used. (defaults to FALSE) (V39)

CYCLE_KIND (multiple state selections):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V37)
 GTCY_Labels (STRPTR *) - Pointer to NULL-terminated array of strings that are the choices offered by the cycle gadget. This tag is required. (V36)
 GTCY_Active (UWORD) - The ordinal number (counting from zero) of the initially active choice of a cycle gadget (defaults to zero). (V36)

INTEGER_KIND (numeric entry):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V36)
 GA_Immediate (BOOL) - Hear IDCMP_GADGETDOWN events from integer gadget (defaults to FALSE). (V39)

GA_TabCycle (BOOL) - 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). (V37)

GTIN_Number (LONG) - The initial contents of the integer gadget (defaults to 0). (V36)

GTIN_MaxChars (UWORD) - The maximum number of digits that the integer gadget is to hold (defaults to 10). (V36)

GTIN_EditHook (struct Hook *) - Hook to use as a custom integer gadget edit hook (StringExtend->EditHook) for this gadget. GadTools will allocate the StringExtend->WorkBuffer for you. (defaults to NULL). (V37)

STRINGA_ExitHelp (BOOL) - Set to TRUE to have the help-key cause an exit from the integer gadget. You will then receive an IDCMP_GADGETUP event with Code = 0x5F (rawkey for help). (defaults to FALSE) (V37)

STRINGA_Justification - Controls the justification of the contents of an integer gadget. Choose one of STRINGLEFT, STRINGRIGHT, or STRINGCENTER (defaults to STRINGLEFT). (V37)

STRINGA_ReplaceMode (BOOL) - If TRUE, this integer gadget is in replace-mode (defaults to FALSE (insert-mode)). (V37)

LISTVIEW_KIND (scrolling list):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V39)

GTLV_Top (WORD) - Top item visible in the listview. This value will be made reasonable if out-of-range (defaults to 0). (V36)

GTLV_MakeVisible (WORD) - Number of an item that should be forced within the visible area of the listview by doing minimal scrolling. This tag overrides GTLV_Top. (V39)

GTLV_Labels (struct List *) - List of nodes whose ln_Name fields are to be displayed in the listview. (V36)

GTLV_ReadOnly (BOOL) - If TRUE, then listview is read-only (defaults to FALSE). (V36)

GTLV_ScrollWidth (UWORD) - Width of scroll bar for listview. Must be greater than zero (defaults to 16). (V36)

GTLV_ShowSelected (struct Gadget *) - NULL to have the currently selected item displayed beneath the listview under V37 or with a highlight bar in V39. If not NULL, this is a pointer to an already-created GadTools STRING_KIND gadget to have an editable display of the currently selected item. If the tag is not present, the currently selected item will not be displayed. (V36)

GTLV_Selected (UWORD) - Ordinal number of currently selected item, or ~0 to have no current selection (defaults to ~0). (V36)

LAYOUTA_Spacing (UWORD) - Extra space to place between lines of listview (defaults to 0). (V36)

GTLV_ItemHeight (UWORD) - The exact height of an item. This is normally useful for listviews that use the GTLV_Callback rendering hook (defaults to ng->ng_TextAttr->ta_YSize). (V39)

GTLV_Callback (struct Hook *) - Callback hook for various listview operations. As of V39, the only callback supported is for custom rendering of individual items in the listview. The call back hook is called with:

A0 - struct Hook *

A1 - struct LVDrawMsg *

A2 - struct Node *

The callback hook `*must*` check the `lvdm_MethodID` field of the message and only do processing if it equals `LV_DRAW`. If any other value is passed, the callback hook must return `LVCB_UNKNOWN`

`GTLV_MaxPen` (UWORD) - The maximum pen number used by rendering in a custom rendering callback hook. This is used to optimize the rendering and scrolling of the listview display (default is the maximum pen number used by all of `TEXTPEN`, `BACKGROUNDPEN`, `FILLPEN`, `TEXTFILLPEN`, and `BLOCKPEN`). (V39)

`MX_KIND` (mutually exclusive, radio buttons):

`GA_Disabled` (BOOL) - Set to `TRUE` to disable gadget, `FALSE` otherwise (defaults to `FALSE`). (V39)

`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. This tag is required. (V36)

`GTMX_Active` (UWORD) - The ordinal number (counting from zero) of the initially active choice of an mx gadget (defaults to 0). (V36)

`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 1). (V36)

`GTMX_Scaled` (BOOL) - If true, then mx gadget imagery will be scaled to fit the gadget's width & height. Otherwise, a fixed size of `MXWIDTH` by `MXHEIGHT` will be used. When setting this tag to `TRUE`, you should typically set the height of the gadget to be `(ng.ng_TextAttr->ta_YSize + 1)`. (defaults to `FALSE`.) (V39)

`GTMX_TitlePlace` - One of `PLACETEXT_LEFT`, `PLACETEXT_RIGHT`, `PLACETEXT_ABOVE`, or `PLACETEXT_BELOW`, indicating where the title of the gadget is to be displayed. Without this tag, the `NewGadget.ng_GadgetText` field is ignored for `MX_KIND` gadgets. (V39)

`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). (V36)

`NUMBER_KIND` (read-only numeric):

`GTNM_Number` (LONG) - A signed long integer to be displayed as a read-only number (defaults to 0). (V36)

`GTNM_Border` (BOOL) - If `TRUE`, this flag asks for a recessed border to be placed around the gadget. (V36)

`GTNM_FrontPen` (UBYTE) - The pen to use when rendering the number (defaults to `DrawInfo->dri_Pens[TEXTPEN]`). (V39)

`GTNM_BackPen` (UBYTE) - The pen to use when rendering the background of the number (defaults to leaving the background untouched). (V39)

`GTNM_Justification` (UBYTE) - Determines how the number is rendered within the gadget box. `GTJ_LEFT` will make the rendering be flush with the left side of the gadget, `GTJ_RIGHT` will make it flush with the right side, and `GTJ_CENTER` will center the number within the gadget box. Under V39, using this tag also required using `{GTNM_Clipped, TRUE}`, otherwise the text would not show up in the gadget. This has been fixed in V40. (defaults to `GTJ_LEFT`). (V39)

`GTNM_Format` (STRPTR) - C-Style formatting string to apply on the number

before display. Be sure to use the 'l' (long) modifier. This string is processed using `exec.library/RawDoFmt()`, so refer to that function for details. (defaults to "%ld") (V39)

GTNM_MaxNumberLen (ULONG) - Maximum number of bytes that can be generated by applying the GTNM_Format formatting string to the number (excluding the NULL terminator). (defaults to 10). (V39)

GTNM_Clippped (BOOL) - Determine whether text should be clipped to the gadget dimensions (defaults to FALSE for gadgets without borders, TRUE for gadgets with borders). (V39)

PALETTE_KIND (color selection):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V36)

GTPA_Depth (UWORD) - Number of bitplanes in the palette (defaults to 1). (V36)

GTPA_Color (UBYTE) - Initially selected color of the palette. This number is a pen number, and not the ordinal color number within the palette gadget itself. (defaults to 1). (V36)

GTPA_ColorOffset (UBYTE) - First color to use in palette (defaults to 0). (V36)

GTPA_IndicatorWidth (UWORD) - The desired width of the current-color indicator, if you want one to the left of the palette. (V36)

GTPA_IndicatorHeight (UWORD) - The desired height of the current-color indicator, if you want one above the palette. (V36)

GTPA_ColorTable (UBYTE *) - Pointer to a table of pen numbers indicating which colors should be used and edited by the palette gadget. This array must contain as many entries as there are colors displayed in the palette gadget. The array provided with this tag must remain valid for the life of the gadget or until a new table is provided. (default is NULL, which causes a 1-to-1 mapping of pen numbers). (V39)

GTPA_NumColors (UWORD) - Number of colors to display in the palette gadget. This override GTPA_Depth and allows numbers which aren't powers of 2. (defaults to 2) (V39)

SCROLLER_KIND (for scrolling through areas or lists):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V36)

GA_RelVerify (BOOL) - Hear every IDCMP_GADGETUP event from scroller (defaults to FALSE). (V36)

GA_Immediate (BOOL) - Hear every IDCMP_GADGETDOWN event from scroller (defaults to FALSE). (V36)

GTSC_Top (WORD) - Top visible in area scroller represents (defaults to 0). (V36)

GTSC_Total (WORD) - Total in area scroller represents (defaults to 0). (V36)

GTSC_Visible (WORD) - Number visible in scroller (defaults to 2). (V36)

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). (V36)

PGA_Freedom - Whether scroller is horizontal or vertical. Choose LORIENT_VERT or LORIENT_HORIZ (defaults to LORIENT_HORIZ). (V36)

SLIDER_KIND (to indicate level or intensity):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V36)

GA_RelVerify (BOOL) - If you want to hear each slider IDCMP_GADGETUP event (defaults to FALSE). (V36)

GA_Immediate (BOOL) - If you want to hear each slider IDCMP_GADGETDOWN event (defaults to FALSE). (V36)

GTSL_Min (WORD) - Minimum level for slider (defaults to 0). (V36)

GTSL_Max (WORD) - Maximum level for slider (defaults to 15). (V36)

GTSL_Level (WORD) - Current level of slider (defaults to 0). (V36)

GTSL_MaxLevelLen (UWORD) - Maximum length in characters of level string when rendered beside slider (defaults to 2). (V36)

GTSL_LevelFormat (STRPTR) - C-Style formatting string for slider level. Be sure to use the 'l' (long) modifier. This string is processed using `exec.library/RawDoFmt()`, so refer to that function for details. (defaults to "%ld"). (V36)

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`). (V36)

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 $(1 \ll n)$ 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. (V36)

GTSL_MaxPixelLen (ULONG) - Indicates the maximum pixel size used up by the level display for any value of the slider. This is mostly useful when dealing with proportional fonts. (defaults to `FontWidth*MaxLevelLen`). (V39)

GTSL_Justification (UBYTE) - Determines how the level display is to be justified within its allotted space. Choose one of `GTJ_LEFT`, `GTJ_RIGHT`, or `GTJ_CENTER` (defaults to `GTJ_LEFT`). (V39)

PGA_Freedom - Set to `LORIENT_VERT` or `LORIENT_HORIZ` to have a vertical or horizontal slider (defaults to `LORIENT_HORIZ`). (V36)

STRING_KIND (text-entry):

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (defaults to FALSE). (V36)

GA_Immediate (BOOL) - Hear IDCMP_GADGETDOWN events from string gadget (defaults to FALSE). (V39)

GA_TabCycle (BOOL) - 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). (V37)

GTST_String (STRPTR) - The initial contents of the string gadget, or NULL (default) if string is to start empty. (V36)

GTST_MaxChars (UWORD) - The maximum number of characters that the string gadget is to hold. (V36)

GTST_EditHook (struct Hook *) - Hook to use as a custom string gadget edit hook (`StringExtend->EditHook`) for this gadget. GadTools will allocate the `StringExtend->WorkBuffer` for you. (defaults to NULL). (V37)

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

STRINGA_Justification - Controls the justification of the contents of

a string gadget. Choose one of `STRINGLEFT`, `STRINGRIGHT`, or `STRINGCENTER` (defaults to `STRINGLEFT`). (V37)

`STRINGA_ReplaceMode` (BOOL) - If `TRUE`, this string gadget is in replace-mode (defaults to `FALSE` (insert-mode)). (V37)

`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`) (V36)

`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`. (V37)

`GTTX_Border` (BOOL) - If `TRUE`, this flag asks for a recessed border to be placed around the gadget. (V36)

`GTTX_FrontPen` (UBYTE) - The pen to use when rendering the text (defaults to `DrawInfo->dri_Pens[TEXTPEN]`). (V39)

`GTTX_BackPen` (UBYTE) - The pen to use when rendering the background of the text (defaults to leaving the background untouched). (V39)

`GTTX_Justification` (UBYTE) - Determines how the text is rendered within the gadget box. `GTJ_LEFT` will make the rendering be flush with the left side of the gadget, `GTJ_RIGHT` will make it flush with the right side, and `GTJ_CENTER` will center the text within the gadget box. Under V39, using this tag also required using `{GTNM_Clippped, TRUE}`, otherwise the text would not show up in the gadget. This has been fixed in V40. (defaults to `GTJ_LEFT`). (V39)

`GTTX_Clippped` (BOOL) - Determine whether text should be clipped to the gadget dimensions (defaults to `FALSE` for gadgets without borders, `TRUE` for gadgets with borders). (V39)

RESULT

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

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 an `IDCMP_GADGETUP` message with a code of `0x09`. Use `{GA_TabCycle, FALSE}` to suppress this.

SEE ALSO

`FreeGadgets()`, `GT_SetGadgetAttrs()`, `GetVisualInfo()`, `<libraries/gadtools.h>`

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 an array of tags providing optional extra parameters, or NULL.

TAGS

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

(Under V39 and higher, this tag also exists for LayoutMenuA() and LayoutMenuItemsA()). (defaults to zero).

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

GTMN_SecondaryError (ULONG *) - Supply a pointer to a NULL-initialized ULONG to receive a descriptive error code. Possible values:
GTMENU_INVALID - NewMenu structure describes an illegal menu. (CreateMenuA() will fail with a NULL result).
GTMENU_TRIMMED - NewMenu structure has too many menus, items, or subitems (CreateMenuA() will succeed, returning a trimmed-down menu structure).

GTMENU_NOMEM - CreateMenuA() ran out of memory. (V37)

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 CreateMenuA() 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).

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

Prior to V39, if you put images into menus using `IM_ITEM` or `IM_SUB` for a `NewMenu->nm_Type`, the image supplied had to be an ordinary struct `Image`. Starting with V39, you can use `boopsi` images.

SEE ALSO

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

1.5 gadtools.library/DrawBevelBoxA

NAME

`DrawBevelBoxA` -- draw a bevelled box. (V36)
`DrawBevelBox` -- varargs stub for `DrawBevelBoxA()`. (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

This function renders a bevelled box of specified dimensions and type into the supplied `RastPort`.

INPUTS

`rport` - `RastPort` into which the box is to be drawn.
`left` - left edge of the box.
`top` - top edge of the box.
`width` - width of the box.
`height` - height of the box.
`tagList` - pointer to an array of tags providing extra parameters

TAGS

`GTBB_Recessed` (BOOL) - Set to anything for a recessed-looking box. If absent, the box defaults, it would be raised. (V36)

`GTBB_FrameType` (ULONG) - Determines what kind of box this function renders. `BBFT_BUTTON` generates a box like what is used around GadTools `BUTTON_KIND` gadgets. `BBFT_RIDGE` generates a box like what is used around GadTools `STRING_KIND` gadgets. Finally, `BBFT_ICONDROPBOX`

generates a box suitable for a standard icon drop box imagery. (defaults to BBFT_BUTTON). (V39)

GT_VisualInfo (APTR) - You MUST supply the value you obtained from an earlier call to GetVisualInfoA() with this tag. (V36)

NOTES

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

SEE ALSO

GetVisualInfoA(), <libraries/gadtools.h>

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.

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().

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(). This value may be NULL.

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. This parameter may be `NULL`, in which case this function fails.
`tagList` - pointer to an array of tags providing optional extra parameters, or `NULL`.

TAGS

There are currently no tags defined for this function.

RESULT

`vi` - pointer to private data, or `NULL` for failure

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.

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;
```

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.

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`. When `NULL`, the message passed in to the function should be sent back to `Intuition` via `ReplyMsg()`

NOTES

Starting with `V39`, this function actually expects and returns pointers to `ExtIntuiMessage` structures, but the prototype was not changed for source code compatibility with older software.

WARNING

If this function returns `NULL`, you must call `ReplyMsg()` on the `IntuiMessage` you passed in to `GT_FilterIMsg()`. That is, if the message was processed by the toolkit you must reply this message to `Intuition` since `gadtools` will not do this automatically.

SEE ALSO

`GT_GetIMsg()`, `GT_PostFilterIMsg()`

1.13 gadtools.library/GT_GetGadgetAttrsA

NAME

`GT_GetGadgetAttrsA` -- request the attributes of a `GadTools` gadget. (`V39`)
`GT_GetGadgetAttrs` -- varargs stub for `GT_GetGadgetAttrsA()`. (`V39`)

SYNOPSIS

```
numProcessed = GT_GetGadgetAttrsA(gad, win, req, taglist)
                        A0  A1  A2  A3
```

```
LONG GT_GetGadgetAttrsA(struct Gadget *, struct Window *,
                        struct Requester *, struct TagItem *);
```

```
numProcessed = GT_GetGadgetAttrs(gad, win, req, firsttag, ...)
```

```
LONG GT_GetGadgetAttrs(struct Gadget *, struct Window *,
                        struct Requester *, Tag, ...);
```

FUNCTION

Retrieve the attributes of the specified gadget, according to the attributes chosen in the tag list. For each entry in the tag list, `ti_Tag` identifies the attribute, and `ti_Data` is a pointer to the long variable where you wish the result to be stored.

INPUTS

`gad` - pointer to the gadget in question. May be `NULL`, in which case this function returns 0

`win` - pointer to the window containing the gadget.

req - reserved for future use, should always be NULL
taglist - pointer to TagItem list.

TAGS

BUTTON_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
FALSE otherwise. (V39)

CHECKBOX_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
FALSE otherwise. (V39)

GTCB_Checked (BOOL) - TRUE if this gadget is currently checked,
FALSE otherwise. (V39)

CYCLE_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
FALSE otherwise. (V39)

GTCY_Active (UWORD) - The ordinal number (counting from zero) of
the active choice of a cycle gadget. (V39)

GTCY_Labels (STRPTR *) - The NULL-terminated array of strings
that are the choices offered by the cycle gadget. (V39)

INTEGER_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
FALSE otherwise. (V39)

GTIN_Number (ULONG) - The contents of the integer gadget. (V39)

LISTVIEW_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
FALSE otherwise. (V39)

GTLV_Top (WORD) - Ordinal number of the top item visible
in the listview. (V39)

GTLV_Labels (struct List *) - The list of nodes whose ln_Name fields
are displayed in the listview. (V39)

GTLV_Selected (UWORD) - Ordinal number of currently selected
item. Returns ~0 if no item is selected. (V39)

MX_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
FALSE otherwise. (V39)

GTMX_Active (UWORD) - The ordinal number (counting from zero) of
the active choice of an mx gadget. (V39)

NUMBER_KIND:

GTNM_Number - The signed long integer that is displayed in
the read-only number. (V39)

PALETTE_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
FALSE otherwise. (V39)

GTPA_Color (UBYTE) - The selected color of the palette. (V39)

GTPA_ColorOffset (UBYTE) - First color used in palette. (V39)

GTPA_ColorTable (UBYTE *) - Pointer to a table of pen numbers
indicating which colors should be used and edited by the palette
gadget. May be NULL, which causes a 1-to-1 mapping of pen numbers.
(V39)

SCROLLER_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
 FALSE otherwise. (V39)
 GTSC_Top (WORD) - Top visible in scroller. (V39)
 GTSC_Total (WORD) - Total in scroller area. (V39)
 GTSC_Visible (WORD) - Number visible in scroller. (V39)

SLIDER_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
 FALSE otherwise. (V39)
 GTSL_Min (WORD) - Minimum level for slider. (V39)
 GTSL_Max (WORD) - Maximum level for slider. (V39)
 GTSL_Level (WORD) - Current level of slider. (V39)

STRING_KIND:

GA_Disabled (BOOL) - TRUE if this gadget is disabled,
 FALSE otherwise. (V39)
 GTST_String (STRPTR) - Returns a pointer to the string gadget's
 buffer. (V39)

TEXT_KIND:

GTTX_Text - Pointer to the string to be displayed in the
 read-only text-display gadget. (V39)

RESULT

numProcessed - the number of attributes successfully filled in.

EXAMPLE

```
long top = 0;
long selected = 0;
long result;
result = GT_GetGadgetAttrs( listview_gad, win, NULL,
    GTLV_Top, &top,
    GTLV_Selected, &selected,
    TAG_DONE );
if ( result != 2 )
{
    printf( "Something's wrong!" );
}
```

WARNING

The pointers you provide within the tag list to store the return values MUST POINT TO LONGWORDS. That is, even if the type of a return value is defined as (UWORD *), you must pass a pointer to a longword of memory.

SEE ALSO

GT_SetGadgetAttrs()

1.14 gadtools.library/GT_GetIMsg

NAME

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

SYNOPSIS

```

img = 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

`img` - pointer to modified `IntuiMessage`, or `NULL` if there are no applicable messages.

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.

Starting with `V39`, this function actually returns a pointer to an `ExtIntuiMessage` structure, but the prototype was not changed for source code compatibility with older software.

SEE ALSO

`GT_ReplyIMsg()`, `GT_FilterIMsg()`

1.15 gadtools.library/GT_PostFilterIMsg

NAME

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

SYNOPSIS

```

img = GT_PostFilterIMsg(modimg)
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(), or NULL in which case this function does nothing and returns NULL

RESULT

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

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.

Starting with V39, this function actually expects and returns pointers to ExtIntuiMessage structures, but the prototype was not changed for source code compatibility with older software.

SEE ALSO

GT_FilterIMsg()

1.16 gadtools.library/GT_RefreshWindow

NAME

GT_RefreshWindow -- refresh all GadTools gadgets in a window. (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/AddGList(), refresh them using intuition.library/RefreshGList(), then call this function.

You should not need this function at other times.

INPUTS

win - pointer to the Window containing GadTools gadgets.
req - reserved for future use, should always be NULL

SEE ALSO

```
GT_BeginRefresh()
```

1.17 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(), or NULL in which case this function does nothing

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.

Starting with V39, this function actually expects a pointer to an ExtIntuiMessage structure, but the prototype was not changed for source code compatibility with older software.

SEE ALSO

GT_GetIMsg()

1.18 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. If an attribute is not provided in the tag list, its value remains unchanged.

INPUTS

gad - pointer to the gadget in question. Starting with V39, this value may be NULL in which case this function does nothing
win - pointer to the window containing the gadget. Starting with V39, this value may be NULL in which case the internal attributes of the gadgets are altered but no rendering occurs.
req - reserved for future use, should always be NULL
tagList - pointer to an array of tags providing optional extra parameters, or NULL.

TAGS

BUTTON_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise. (V36)

CHECKBOX_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise. (V36)

GTCB_Checked (BOOL) - State of checkbox. (V36)

CYCLE_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V37)

GTCY_Active (UWORD) - The ordinal number (counting from zero) of the active choice of a cycle gadget. (V36)

GTCY_Labels (STRPTR *) - Pointer to NULL-terminated array of strings that are the choices offered by the cycle gadget. (V37)

INTEGER_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V36)

GTIN_Number (ULONG) - New value of the integer gadget (V36)

LISTVIEW_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V39)

GTLV_Top (WORD) - Top item visible in the listview. This value will be made reasonable if out-of-range. (V36)

GTLV_MakeVisible (WORD) - Number of an item that should be forced within the visible area of the listview by doing minimal scrolling. This tag overrides GTLV_Top. (V39)

GTLV_Labels (struct List *) - List of nodes 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}. (V36)

GTLV_Selected (UWORD) - Ordinal number of currently selected item. Starting with V39, you can provide ~0 to cause the currently

selected item to be deselected. (V36)

MX_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V39)

GTMX_Active (UWORD) - The ordinal number (counting from zero) of the active choice of an mx gadget. (V36)

NUMBER_KIND:

GTNM_Number (LONG) - A signed long integer to be displayed in the number gadget. (V36)

GTNM_FrontPen (UBYTE) - The pen to use when rendering the number. (V39)

GTNM_BackPen (UBYTE) - The pen to use when rendering the background of the number. (V39)

GTNM_Justification (UBYTE) - Determines how the number is rendered within the gadget box. GTJ_LEFT will make the rendering be flush with the left side of the gadget, GTJ_RIGHT will make it flush with the right side, and GTJ_CENTER will center the number within the gadget box. (V39)

GTNM_Format (STRPTR) - C-Style formatting string to apply on the number before display. Be sure to use the 'l' (long) modifier. This string is processed using exec.library/RawDoFmt(), so refer to that function for details. (V39)

PALETTE_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V36)

GTPA_Color (UBYTE) - Selected color of the palette. This number is a pen number, and not the ordinal color number within the palette gadget itself. (V36)

GTPA_ColorOffset (UBYTE) - First color to use in palette (V39)

GTPA_ColorTable (UBYTE *) - Pointer to a table of pen numbers indicating which colors should be used and edited by the palette gadget. This array must contain as many entries as there are colors displayed in the palette gadget. The array provided with this tag must remain valid for the life of the gadget or until a new table is provided. A NULL table pointer causes a 1-to-1 mapping of pen numbers. (V39)

SCROLLER_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V36)

GTSC_Top (WORD) - Top visible in scroller. (V36)

GTSC_Total (WORD) - Total in scroller area. (V36)

GTSC_Visible (WORD) - Number visible in scroller. (V36)

SLIDER_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V36)

GTSL_Min (WORD) - Minimum level for slider. (V36)

GTSL_Max (WORD) - Maximum level for slider. (V36)

GTSL_Level (WORD) - Current level of slider. (V36)

GTSL_LevelFormat (STRPTR) - C-Style formatting string for slider level. Be sure to use the 'l' (long) modifier. This string is processed using exec.library/RawDoFmt(), so refer to that function for details. (V39)

GTSL_DispatchFunc (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 $(1 \ll n)$ function here. 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. (V39)

GTSL_Justification (UBYTE) - Determines how the level display is to be justified within its allotted space. Choose one of GTJ_LEFT, GTJ_RIGHT, or GTJ_CENTER. (V39)

STRING_KIND:

GA_Disabled (BOOL) - Set to TRUE to disable gadget, FALSE otherwise (V36)

GTST_String (STRPTR) - New contents of the string gadget, or NULL to reset the gadget to the empty state. (V36)

TEXT_KIND:

GTTX_Text - New NULL terminated string to be displayed in the text gadget. NULL means to clear the gadget. (V36)

GTTX_FrontPen (UBYTE) - The pen to use when rendering the text. (V39)

GTTX_BackPen (UBYTE) - The pen to use when rendering the background of the text. (V39)

GTTX_Justification (UBYTE) - Determines how the text is rendered within the gadget box. GTJ_LEFT will make the rendering be flush with the left side of the gadget, GTJ_RIGHT will make it flush with the right side, and GTJ_CENTER will center the text within the gadget box. (V39)

NOTES

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

SEE ALSO

GT_GetGadgetAttrs()

1.19 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 MenuItem's 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 an array of tags providing optional extra parameters, or NULL.

TAGS

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. (V36)

For the following tags, please see the description under LayoutMenuA(). Their behavior is identical when used in LayoutMenuItemsA().

GTMN_TextAttr
 GTMN_NewLookMenus
 GTMN_Checkmark
 GTMN_AmigaKey
 GTMN_FrontPen

RESULT

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

BUGS

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

SEE ALSO

CreateMenuA(), GetVisualInfoA()

1.20 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 LayoutMenuA(struct Menu *, APTR, struct TagItem *);
```

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

```
BOOL LayoutMenu(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 an array of tags providing optional extra parameters, or NULL.

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. (V36)
GTMN_NewLookMenus (BOOL) - If you ask for WA_NewLookMenus for your window, you should ask for this tag as well. This informs GadTools to use the appropriate checkmark, Amiga-key, and colors. (V39)
GTMN_Checkmark (struct Image *) - If you are using a custom image for a checkmark (WA_Checkmark), also pass it to GadTools, so it can lay the menus out accordingly. (V39)
GTMN_AmigaKey (struct Image *) - If you are using a custom image for the Amiga-key in menus (WA_AmigaKey), also pass it to GadTools, so it can lay the menus out accordingly. (V39)
GTMN_FrontPen (ULONG) - This tag has existed for CreateMenus(), but now LayoutMenusA() has it too. If a legitimate pen number is supplied, it is used for coloring the menu items (in preference to anything passed to CreateMenus()). If GTMN_NewLookMenus has been specified, this tag defaults to using the screen's BARDETAILPEN, else it defaults to "do nothing". For visual consistency, we recommend you omit this tag in all functions, and let the defaults be used. (V39)

RESULT

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

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()