

**intuition**

<b>COLLABORATORS</b>
----------------------

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

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>intuition</b>	<b>1</b>
1.1	intuition.doc . . . . .	1
1.2	intuition.library/ActivateGadget . . . . .	2
1.3	intuition.library/ActivateWindow . . . . .	3
1.4	intuition.library/AddClass . . . . .	4
1.5	intuition.library/AddGadget . . . . .	4
1.6	intuition.library/AddGLList . . . . .	6
1.7	intuition.library/AllocRemember . . . . .	7
1.8	intuition.library/AutoRequest . . . . .	8
1.9	intuition.library/BeginRefresh . . . . .	10
1.10	intuition.library/BuildEasyRequestArgs . . . . .	12
1.11	intuition.library/BuildSysRequest . . . . .	13
1.12	intuition.library/ChangeWindowBox . . . . .	16
1.13	intuition.library/ClearDMRequest . . . . .	17
1.14	intuition.library/ClearMenuStrip . . . . .	18
1.15	intuition.library/ClearPointer . . . . .	18
1.16	intuition.library/CloseScreen . . . . .	19
1.17	intuition.library/CloseWindow . . . . .	20
1.18	intuition.library/CloseWorkBench . . . . .	22
1.19	intuition.library/CurrentTime . . . . .	23
1.20	intuition.library/DisplayAlert . . . . .	24
1.21	intuition.library/DisplayBeep . . . . .	25
1.22	intuition.library/DisposeObject . . . . .	26
1.23	intuition.library/DoubleClick . . . . .	27
1.24	intuition.library/DrawBorder . . . . .	27
1.25	intuition.library/DrawImage . . . . .	28
1.26	intuition.library/DrawImageState . . . . .	29
1.27	intuition.library/EasyRequestArgs . . . . .	30
1.28	intuition.library/EndRefresh . . . . .	33
1.29	intuition.library/EndRequest . . . . .	34

---

1.30	intuition.library/EraseImage . . . . .	34
1.31	intuition.library/FreeClass . . . . .	35
1.32	intuition.library/FreeRemember . . . . .	36
1.33	intuition.library/FreeScreenDrawInfo . . . . .	37
1.34	intuition.library/FreeSysRequest . . . . .	38
1.35	intuition.library/GadgetMouse . . . . .	39
1.36	intuition.library/GetAttr . . . . .	39
1.37	intuition.library/GetDefaultPubScreen . . . . .	40
1.38	intuition.library/GetDefPrefs . . . . .	41
1.39	intuition.library/GetPrefs . . . . .	42
1.40	intuition.library/GetScreenData . . . . .	42
1.41	intuition.library/GetScreenDrawInfo . . . . .	44
1.42	intuition.library/InitRequester . . . . .	45
1.43	intuition.library/IntuiTextLength . . . . .	46
1.44	intuition.library/ItemAddress . . . . .	47
1.45	intuition.library/LockIBase . . . . .	47
1.46	intuition.library/LockPubScreen . . . . .	48
1.47	intuition.library/LockPubScreenList . . . . .	49
1.48	intuition.library/MakeClass . . . . .	50
1.49	intuition.library/MakeScreen . . . . .	52
1.50	intuition.library/ModifyIDCMP . . . . .	53
1.51	intuition.library/ModifyProp . . . . .	54
1.52	intuition.library/MoveScreen . . . . .	55
1.53	intuition.library/MoveWindow . . . . .	56
1.54	intuition.library/MoveWindowInFrontOf . . . . .	57
1.55	intuition.library/NewModifyProp . . . . .	57
1.56	intuition.library/NewObject . . . . .	58
1.57	intuition.library/NextObject . . . . .	59
1.58	intuition.library/NextPubScreen . . . . .	60
1.59	intuition.library/ObtainGIRPort . . . . .	61
1.60	intuition.library/OffGadget . . . . .	61
1.61	intuition.library/OffMenu . . . . .	62
1.62	intuition.library/OnGadget . . . . .	63
1.63	intuition.library/OnMenu . . . . .	64
1.64	intuition.library/OpenScreen . . . . .	64
1.65	intuition.library/OpenScreenTagList . . . . .	71
1.66	intuition.library/OpenWindow . . . . .	72
1.67	intuition.library/OpenWindowTagList . . . . .	85
1.68	intuition.library/OpenWorkBench . . . . .	86

---

1.69	intuition.library/PointInImage	87
1.70	intuition.library/PrintIText	88
1.71	intuition.library/PubScreenStatus	88
1.72	intuition.library/QueryOverscan	89
1.73	intuition.library/RefreshGadgets	90
1.74	intuition.library/RefreshGList	91
1.75	intuition.library/RefreshWindowFrame	92
1.76	intuition.library/ReleaseGIRPort	93
1.77	intuition.library/RemakeDisplay	93
1.78	intuition.library/RemoveClass	94
1.79	intuition.library/RemoveGadget	95
1.80	intuition.library/RemoveGList	96
1.81	intuition.library/ReportMouse	96
1.82	intuition.library/Request	98
1.83	intuition.library/ResetMenuStrip	99
1.84	intuition.library/RethinkDisplay	100
1.85	intuition.library/ScreenToBack	101
1.86	intuition.library/ScreenToFront	101
1.87	intuition.library/SetAttrsA	102
1.88	intuition.library/SetDefaultPubScreen	102
1.89	intuition.library/SetDMRequest	103
1.90	intuition.library/SetEditHook	104
1.91	intuition.library/SetGadgetAttrsA	104
1.92	intuition.library/SetMenuStrip	105
1.93	intuition.library/SetMouseQueue	106
1.94	intuition.library/SetPointer	107
1.95	intuition.library/SetPrefs	108
1.96	intuition.library/SetPubScreenModes	109
1.97	intuition.library/SetWindowTitles	109
1.98	intuition.library/ShowTitle	110
1.99	intuition.library/SizeWindow	111
1.100	intuition.library/SysReqHandler	112
1.101	intuition.library/UnlockIBase	115
1.102	intuition.library/UnlockPubScreen	115
1.103	intuition.library/UnlockPubScreenList	116
1.104	intuition.library/ViewAddress	116
1.105	intuition.library/ViewPortAddress	117
1.106	intuition.library/WBenchToBack	117
1.107	intuition.library/WBenchToFront	118
1.108	intuition.library/WindowLimits	119
1.109	intuition.library/WindowToBack	120
1.110	intuition.library/WindowToFront	120
1.111	intuition.library/ZipWindow	121

# Chapter 1

## intuition

### 1.1 intuition.doc

ActivateGadget ()	NextObject ()
ActivateWindow ()	NextPubScreen ()
AddClass ()	ObtainGIRPort ()
AddGadget ()	OffGadget ()
AddGLList ()	OffMenu ()
AllocRemember ()	OnGadget ()
AutoRequest ()	OnMenu ()
BeginRefresh ()	OpenScreen ()
BuildEasyRequestArgs ()	OpenScreenTagList ()
BuildSysRequest ()	OpenWindow ()
ChangeWindowBox ()	OpenWindowTagList ()
ClearDMRequest ()	OpenWorkBench ()
ClearMenuStrip ()	PointInImage ()
ClearPointer ()	PrintIText ()
CloseScreen ()	PubScreenStatus ()
CloseWindow ()	QueryOverscan ()
CloseWorkBench ()	RefreshGadgets ()
CurrentTime ()	RefreshGLList ()
DisplayAlert ()	RefreshWindowFrame ()
DisplayBeep ()	ReleaseGIRPort ()
DisposeObject ()	RemakeDisplay ()
DoubleClick ()	RemoveClass ()
DrawBorder ()	RemoveGadget ()
DrawImage ()	RemoveGLList ()
DrawImageState ()	ReportMouse ()
EasyRequestArgs ()	Request ()
EndRefresh ()	ResetMenuStrip ()
EndRequest ()	RethinkDisplay ()
EraseImage ()	ScreenToBack ()
FreeClass ()	ScreenToFront ()
FreeRemember ()	SetAttrsA ()
FreeScreenDrawInfo ()	SetDefaultPubScreen ()
FreeSysRequest ()	SetDMRequest ()
GadgetMouse ()	SetEditHook ()
GetAttr ()	SetGadgetAttrsA ()
GetDefaultPubScreen ()	SetMenuStrip ()
GetDefPrefs ()	SetMouseQueue ()
GetPrefs ()	SetPointer ()

GetScreenData()	SetPrefs()
GetScreenDrawInfo()	SetPubScreenModes()
InitRequester()	SetWindowTitles()
IntuiTextLength()	ShowTitle()
ItemAddress()	SizeWindow()
LockIBase()	SysReqHandler()
LockPubScreen()	UnlockIBase()
LockPubScreenList()	UnlockPubScreen()
MakeClass()	UnlockPubScreenList()
MakeScreen()	ViewAddress()
ModifyIDCMP()	ViewPortAddress()
ModifyProp()	WBenchToBack()
MoveScreen()	WBenchToFront()
MoveWindow()	WindowLimits()
MoveWindowInFrontOf()	WindowToBack()
NewModifyProp()	WindowToFront()
NewObject()	ZipWindow()

## 1.2 intuition.library/ActivateGadget

### NAME

ActivateGadget -- Activate a (string or custom) gadget.

### SYNOPSIS

```
Success = ActivateGadget( Gadget, Window, Request )
D0                A0        A1        A2
```

```
BOOL ActivateGadget( struct Gadget *, struct Window *,
                    struct Requester * );
```

### FUNCTION

Activates a string or custom gadget. If successful, this means that the user does not need to click in the gadget before typing.

The window parameter must point to the window which contains the gadget. If the gadget is actually in a requester, the window must contain the requester, and a pointer to the requester must also be passed. The requester parameter must only be valid if the gadget has the GTYP\_REQGADGET flag set, a requirement for all requester gadgets.

The success of this function depends on a rather complex set of conditions. The intent is that the user is never interrupted from what interactions he may have underway.

The current set of conditions includes:

- The window must be active. If you are opening a new window and want an active gadget in it, it is not sufficient to assume that the WFLG\_ACTIVATE flag has taken effect by the time OpenWindow() returns, even if you insert a delay of some finite amount of time. Use the IDCMP\_ACTIVEWINDOW IntuiMessage to tell when your window really becomes active. Many programs use an event loop that calls ActivateGadget() whenever they receive the IDCMP\_ACTIVEWINDOW message, and also the IDCMP\_MOUSEBUTTONS messages, and so on, to keep the

gadget active until it is used (or the user selects some other "Cancel" gadget).

- No other gadgets may be in use. This includes system gadgets, such as those for window sizing, dragging, etc.
- If the gadget is in a requester, that requester must be active. (Use IDCMP\_REQSET and IDCMP\_REQCLEAR).
- The right mouse button cannot be held down (e.g. menus)

NOTE: Don't try to activate a gadget which is disabled or not attached to a window or requester.

#### INPUTS

Gadget = pointer to the gadget that you want activated.  
 Window = pointer to a window structure containing the gadget.  
 Requester = pointer to a requester (may be NULL if this isn't a requester gadget (i.e. GTYP\_REQGADGET is not set)).

#### RESULT

If the conditions above are met, and the gadget is in fact a string gadget, then this function will return TRUE, else FALSE.

#### BUGS

At present, this function will not return FALSE if a custom gadget declines to be activated.

#### SEE ALSO

## 1.3 intuition.library/ActivateWindow

#### NAME

ActivateWindow -- Activate an Intuition window.

#### SYNOPSIS

```
[success =] ActivateWindow( Window )
[D0]                                A0

[LONG] ActivateWindow( struct Window * );
/* returns LONG in V36 and higher */
```

#### FUNCTION

Activates an Intuition window.

Note that this call may have its action deferred: you cannot assume that when this call is made the selected window has become active. This action will be postponed while the user plays with gadgets and menus, or sizes and drags windows. You may detect when the window actually has become active by the IDCMP\_ACTIVEWINDOW IDCMP message.

This call is intended to provide flexibility but not to confuse the user. Please call this function synchronously with some action by the user.

#### INPUTS

Window = a pointer to a Window structure

---



## RESULT

V35 and before: None.

V36 and later: returns zero if no problem queuing up the request for deferred action

## BUGS

Calling this function in a tight loop can blow out Intuition's deferred action queue.

## SEE ALSO

OpenWindow(), and the WFLG\_ACTIVATE window flag

## 1.4 intuition.library/AddClass

## NAME

AddClass -- Make a public class available (V36)

## SYNOPSIS

```
AddClass( Class )
          A0
```

```
VOID AddClass( struct IClass * );
```

## FUNCTION

Adds a public boopsi class to the internal list of classes available for public consumption.

You must call this function after you call MakeClass().

## INPUTS

Class = pointer returned by MakeClass()

## RESULT

Nothing returned.

## NOTES

## BUGS

Although there is some protection against creating classes with the same name as an existing class, this function does not do any checking or other dealings with like-named classes. Until this is rectified, only officially registered names can be used for public classes, and there is no "class replacement" policy in effect.

## SEE ALSO

MakeClass(), FreeClass(), RemoveClass()  
Document "Basic Object-Oriented Programming System for Intuition"  
and the "boopsi Class Reference" document.

## 1.5 intuition.library/AddGadget

## NAME

AddGadget -- Add a gadget to the gadget list of a window.

## SYNOPSIS

```
RealPosition = AddGadget( Window, Gadget, Position )
```

```
D0                      A0      A1      D0
```

```
UWORD AddGadget( struct Window *, struct Gadget *, UWORD );
```

## FUNCTION

Adds the specified gadget to the gadget list of the given window, linked in at the position in the list specified by the position argument (that is, if Position == 0, the gadget will be inserted at the head of the list, and if Position == 1 then the gadget will be inserted after the first gadget and before the second). If the position you specify is greater than the number of gadgets in the list, your gadget will be added to the end of the list.

Calling AddGadget() does not cause your gadget to be redisplayed. The benefit of this is that you may add several gadgets without having the gadget list redrawn every time.

This procedure returns the position at which your gadget was added.

NOTE: A relatively safe way to add the gadget to the end of the list is to specify a position of -1 (i.e., (UWORD) ~0). That way, only the 65536th (and multiples of it) will be inserted at the wrong position. The return value of the procedure will tell you where it was actually inserted.

NOTE: The system window gadgets are initially added to the front of the gadget list. The reason for this is: If you position your own gadgets in some way that interferes with the graphical representation of the system gadgets, the system's ones will be "hit" first by user. If you then start adding gadgets to the front of the list, you will disturb this plan, so beware. On the other hand, if you don't violate the design rule of never overlapping your gadgets, there's no problem.

NOTE: You may not add your own gadgets to a screen. Gadgets may be added to backdrop windows, however, which can be visually similar, but also provide an IDCMP channel for gadget input messages.

## INPUTS

Window = pointer to the window to get your gadget

Gadget = pointer to the new gadget

Position = integer position in the list for the new gadget (starting from zero as the first position in the list)

## RESULT

Returns the position of where the gadget was actually added.

## BUGS

## SEE ALSO

AddGList(), RemoveGadget(), RemoveGList()

## 1.6 intuition.library/AddGList

### NAME

AddGList -- Add a linked list of gadgets to a window or requester.

### SYNOPSIS

```
RealPosition = AddGList( Window, Gadget, Position, Numgad, Requester )
D0                A0      A1      D0      D1      A2
```

```
UWORD AddGList( struct Window *, struct Gadget *, UWORD, WORD,
                struct Requester * );
```

### FUNCTION

Adds the list of gadgets to the gadget list of the given window or requester linked in at the position in the list specified by the position argument.

See AddGadget() for more information about gadget list position, and more information about gadgets in general.

The requester parameter will be ignored unless the GTYP\_REQGADGET bit is set in the GadgetType field of the first gadget in the list. In that case, the gadget list is added to the requester gadgets. NOTE: be sure that GTYP\_REQGADGET is either set or cleared consistently for all gadgets in the list. NOTE ALSO: The window parameter should point to the window that the requester (will) appear in.

Will add 'Numgad' gadgets from gadget list linked by the field NextGadget, or until some NextGadget field is found to be NULL. Does not assume that the Numgad'th gadget has NextGadget equal to NULL.

NOTE WELL: In order to link your gadget list in, the NextGadget field of the Numgad'th (or last) gadget will be modified. Thus, if you are adding the first 3 gadgets from a linked list of five gadgets, this call will sever the connection between your third and fourth gadgets.

### INPUTS

Window = pointer to the window to get your gadget  
 Gadget = pointer to the first gadget to be added  
 Position = integer position in the list for the new gadget  
           (starting from zero as the first position in the list)  
 Numgad = the number of gadgets from the linked list to be added  
           if Numgad equals -1, the entire null-terminated list of  
           gadgets will be added.  
 Requester = the requester the gadgets will be added to if the  
           GTYP\_REQGADGET GadgetType flag is set for the first gadget  
           in the list

### RESULT

Returns the position of where the first gadget in the list was actually added.

### BUGS

### SEE ALSO

AddGadget(), RemoveGadget(), RemoveGList()

## 1.7 intuition.library/AllocRemember

### NAME

AllocRemember -- AllocMem() with tracking to make freeing easy.

### SYNOPSIS

```
MemBlock = AllocRemember( RememberKey, Size, Flags )
D0                      A0                      D0      D1
```

```
APTR AllocRemember( struct Remember **, ULONG, ULONG );
```

### FUNCTION

This routine calls the Exec AllocMem() function for you, but also links the parameters of the allocation into a master list, so that you can simply call the Intuition routine FreeRemember() at a later time to deallocate all allocated memory without being required to remember the details of the memory you've allocated.

This routine will have two primary uses:

- Let's say that you're doing a long series of allocations in a procedure. If any one of the allocations fails, your program may need to abort the procedure. Abandoning ship correctly involves freeing up what memory you've already allocated. This procedure allows you to free up that memory easily, without being required to keep track of how many allocations you've already done, what the sizes of the allocations were, or where the memory was allocated.
- Also, in the more general case, you may do all of the allocations in your entire program using this routine. Then, when your program is exiting, you can free it all up at once with a simple call to FreeRemember().

You create the "anchor" for the allocation master list by creating a variable that's a pointer to struct Remember, and initializing that pointer to NULL. This is called the RememberKey. Whenever you call AllocRemember(), the routine actually does two memory allocations, one for the memory you want and the other for a copy of a Remember structure. The Remember structure is filled in with data describing your memory allocation, and it's linked into the master list pointed to by your RememberKey. Then, to free up any memory that's been allocated, all you have to do is call FreeRemember() with your RememberKey.

Please read the FreeRemember() function description, too. As you will see, you can select either to free just the link nodes and keep all the allocated memory for yourself, or to free both the nodes and your memory buffers.

### INPUTS

RememberKey = the address of a pointer to struct Remember. Before the very first call to AllocRemember, initialize this pointer to NULL.

Size = the size in bytes of the memory allocation. Please refer to the `exec.library/AllocMem()` function for details.

Flags = the specifications for the memory allocation. Please refer to the `exec.library/AllocMem()` function for details.

**EXAMPLE**

```
struct Remember *RememberKey;
RememberKey = NULL;
buffer = AllocRemember(&RememberKey, BUFSIZE, MEMF_CHIP);
if (buffer)
{
    /* Use the buffer */
    ...
}
FreeRemember(&RememberKey, TRUE);
```

**RESULT**

If the memory allocation is successful, this routine returns the byte address of your requested memory block. Also, the node to your block will be linked into the list pointed to by your `RememberKey` variable. If the allocation fails, this routine returns `NULL` and the list pointed to by `RememberKey`, if any, will be unchanged.

**BUGS**

This function makes two allocations for each memory buffer you request. This is neither fast nor good for memory fragmentation.

This function should use the `exec AllocPool()` function internally, at least for the `Remember` headers.

**SEE ALSO**

`FreeRemember()`, `exec.library/AllocMem()`

## 1.8 intuition.library/AutoRequest

**NAME**

`AutoRequest` -- Automatically build and get response from a requester.

**SYNOPSIS**

```
Response = AutoRequest( Window, BodyText, PosText, NegText,
D0                      A0      A1      A2      A3
                      PosFlags, NegFlags, Width, Height )
D0                      D1      D2      D3
```

```
BOOL AutoRequest( struct Window *, struct IntuiText *,
                  struct IntuiText *, struct IntuiText *, WORD, WORD );
```

**FUNCTION**

This procedure automatically builds a requester for you and then waits for a response from the user, or for the system to satisfy your request. If the response is positive, this procedure returns `TRUE`. If the response is negative, this procedure returns `FALSE`.

An `IDCMPFlag` specification is created by bitwise "or'ing" your

PosFlags, NegFlags, and the IDCMP classes IDCMP\_GADGETUP and IDCMP\_RAWKEY. You may specify zero flags for either the PosFlags or NegFlags arguments.

The IntuiText arguments, and the width and height values, are passed directly to the BuildSysRequest() procedure along with your window pointer and the IDCMP flags. Please refer to BuildSysRequest() for a description of the IntuiText that you are expected to supply when calling this routine. It's an important but long-winded description that need not be duplicated here.

If the BuildSysRequest() procedure does not return a pointer to a window, it will return TRUE or FALSE (not valid structure pointers) instead, and these BOOL values will be returned to you immediately.

On the other hand, if a valid window pointer is returned, that window will have had its IDCMP ports and flags initialized according to your specifications. AutoRequest() then waits for IDCMP messages on the UserPort, which satisfies one of four requirements:

- either the message is of a class that matches one of your PosFlags arguments (if you've supplied any), in which case this routine returns TRUE. Or
- the message class matches one of your NegFlags arguments (if you've supplied any), in which case this routine returns FALSE. Or
- the IDCMP message is of class IDCMP\_GADGETUP, which means that one of the two gadgets, as provided with the PosText and NegText arguments, was selected by the user. If the TRUE gadget was selected, TRUE is returned. If the FALSE gadget was selected, FALSE is returned.
- Lastly, two IDCMP\_RAWKEY messages may satisfy the request: those for the V and B keys with the left Amiga key depressed. These keys, satisfy the gadgets on the left or right side of the requester--TRUE or FALSE--, respectively.

NOTE: For V36, these two keys left-Amiga-B and V are processed through the default keymap.

When the dust has settled, this routine calls FreeSysRequest() if necessary to clean up the requester and any other allocated memory.

NOTE: For V36, this function now switches the processor stack to ensure sufficient stack space for the function to succeed.

#### INPUTS

Window = pointer to a Window structure. See BuildSysRequest() for a full discussion.  
BodyText = pointer to an IntuiText structure  
PosText = pointer to an IntuiText structure, may be NULL.  
NegText = pointer to an IntuiText structure, MUST be valid!  
PosFlags = flags for the IDCMP  
NegFlags = flags for the IDCMP  
Width, Height = the sizes to be used for the rendering of the requester

NOTE for V36: The width and height parameters are ignored, as are several other specifications in the IntuiText, to make

---

`AutoRequest()` requesters retroactively conform to the new look designed for `EasyRequest()`.

#### RESULT

The return value is either `TRUE` or `FALSE`. See the text above for a complete description of the chain of events that might lead to either of these values being returned.

#### BUGS

The requester no longer devolves into a call to `DisplayAlert()` if there is not enough memory for the requester.

#### SEE ALSO

`EasyRequest()`, `BuildSysRequest()`, `SysReqHandler()`

## 1.9 intuition.library/BeginRefresh

#### NAME

`BeginRefresh` -- Sets up a window for optimized refreshing.

#### SYNOPSIS

```
BeginRefresh( Window )
            A0
```

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

#### FUNCTION

This routine sets up your window for optimized refreshing.

Its role is to provide Intuition integrated access to the Layers library function `BeginUpdate()`. Its additional contribution is to be sure that locking protocols for layers are followed, by locking both layers of a `WFLG_GIMMEZEROZERO` window only after the parent `Layer_Info` has been locked. Also, the `WFLG_WINDOWREFRESH` flag is set in your window, for your information.

The purpose of `BeginUpdate()`, and hence `BeginRefresh()`, is to restrict rendering in a window (layer) to the region that needs refreshing after an operation such as window sizing or uncovering. This restriction to the "damage region" persists until you call `EndRefresh()`.

For instance, if you have a `WFLG_SIMPLE_REFRESH` window which is partially concealed and the user brings it to the front, you can receive an `IDCMP_REFRESHWINDOW` message asking you to refresh your display. If you call `BeginRefresh()` before doing any of the rendering, then the layer that underlies your window will be arranged so that the only rendering that will actually take place will be that which goes to the newly-revealed areas. This is very performance-efficient, and visually attractive.

After you have performed your refresh of the display, you should call `EndRefresh()` to reset the state of the layer and the window. Then you may proceed with rendering to the entire window as usual.

---

You learn that your window needs refreshing by receiving either a message of class IDCMP\_REFRESHWINDOW through the IDCMP, or an input event of class IECLASS\_REFRESHWINDOW through the Console device. Whenever you are told that your window needs refreshing, you should call `BeginRefresh()` and `EndRefresh()` to clear the refresh-needed state, even if you don't plan on doing any rendering. You may relieve yourself of even this burden by setting the `WFLG_NOCAREREFRESH` flag when opening your window.

**WARNING:** You should only perform graphics refreshing operations during the period between calling `BeginRefresh()` and `EndRefresh()`. In particular, do not call `RefreshGadgets()` or `RefreshGList()`, since the locking protocol internal to Intuition runs the risk of creating a deadlock. Note that Intuition refreshes the gadgets (through the damage region) before it sends the IDCMP\_REFRESHWINDOW message.

**ANOTHER WARNING:** The concept of multiple refresh passes using `EndRefresh( w, FALSE )` is not completely sound without further protection. The reason is that between two sessions, more damage can occur to your window. Your final `EndRefresh( w, TRUE )` will dispose of all damage, including the new, and your initial refreshing pass will never get the chance to refresh the new damage.

To avoid this, you must protect your session using `LockLayerInfo()` which will prevent Intuition from performing window operations or anything else which might cause further damage from occurring. Again, while holding the `LayerInfo` lock make no Intuition function calls dealing with gadgets; just render.

You can, however, call `InstallClipRegion()` for the different refresh passes, if you have two clip regions.

**SIMILAR WARNING:** Your program and Intuition "share" your window layer's `DamageList`. `BeginRefresh()` helps arbitrate this sharing, but the lower-level function `layers.library/BeginUpdate()` does not. It isn't really supported to use `BeginUpdate()` on a window's layer, but if you do--for whatever reason--it is critical that you first acquire the `LayerInfo` lock as in the above example: even if you only have one pass of refresh rendering to do. Otherwise, the refreshing of your window's borders and gadgets can be incomplete, and the problem might occur only under certain conditions of task priority and system load.

#### EXAMPLE

Code fragment for "two pass" window refreshing, in response to an IDCMP\_REFRESHWINDOW message:

```
switch ( imsg->Class )
{
    ...
    case IDCMP_REFRESHWINDOW:
        window = imsg->IDCMPWindow;

        /* this lock only needed for "two-pass" refreshing */
        LockLayerInfo( &window->WScreen->LayerInfo );
```



```

/* refresh pass for region 1 */
origclip = InstallClipRegion( window->WLayer, region1 );
BeginRefresh( window );
myRefreshRegion1( window );
EndRefresh( window, FALSE );

/* refresh pass for region 2 */
InstallClipRegion( window->WLayer, region2 );
BeginRefresh( window );
myRefreshRegion2( window );
EndRefresh( window, TRUE );          /* and dispose damage list */

/* restore and unlock */
InstallClipRegion( window->WLayer, origclip );
UnlockLayerInfo( &window->WScreen->LayerInfo );
break;
...
}

```

#### INPUTS

Window = pointer to the window structure which needs refreshing

#### RESULT

None

#### BUGS

This function should check the return code of layers.library/BeginUpdate(), and abort if that function fails.

#### SEE ALSO

EndRefresh(), layers.library/BeginUpdate(), OpenWindow()  
 layer.library/InstallClipRegion(), graphics.library/LockLayerInfo()  
 The "Windows" chapter of the Intuition Reference Manual

## 1.10 intuition.library/BuildEasyRequestArgs

#### NAME

BuildEasyRequestArgs -- Simple creation of system request. (V36)  
 BuildEasyRequest -- Varargs stub for BuildEasyRequestArgs(). (V36)

#### SYNOPSIS

```

ReqWindow = BuildEasyRequestArgs( RefWindow, easyStruct, IDCMP, Args )
D0                                     A0          A1          D0          A3

struct Window *BuildEasyRequestArgs( struct Window *,
                                     struct EasyStruct *, ULONG, APTR );

ReqWindow = BuildEasyRequest( RefWindow, easyStruct, IDCMP, Arg1, ... )

struct Window *BuildEasyRequest( struct Window *,
                                 struct EasyStruct *, ULONG, APTR, ... );

```

#### FUNCTION

This function is to EasyRequest() as BuildSysRequest() is to AutoRequest(): it returns a pointer to the system requester

window. The input from that window can then be processed under application control.

It is recommended that this processing be done with `SysReqHandler()`, so that future enhancement to the processing will be enjoyed.

After you have determined that the requester is satisfied or cancelled, you must free this requester using `FreeSysRequest()`.

Please see the autodoc for `EasyRequest()`.

NOTE: This function switches the processor stack to ensure sufficient stack space for the function to complete.

#### INPUTS

`Window` = reference window for requester: determines the requester window title and screen.  
`easyStruct` = pointer to `EasyStruct` structure, as described in the `EasyRequest()` autodocs.  
`IDCMP` = (NOT A POINTER) provided application specific IDCMP flags for the system requester window.  
`Args` = see `EasyRequest()`

#### RESULT

A pointer to the system request window opened. In the event of problems, you may also be returned the value '0' which is to be interpreted as the "FALSE, Cancel" choice, or (if you have a second gadget defined) the value '1', which is to be taken to mean the equivalent of your corresponding left-most gadget.

If there is a problem creating the window, a recoverable alert may be substituted for the requester, and the result, either 0 or 1, returned.

#### BUGS

Does not put up alternative alert.  
 See also BUGS listed for `EasyRequestArgs()`.

#### SEE ALSO

`EasyRequestArgs()`, `FreeSysRequest()`, `SysReqHandler()`,  
`BuildSysRequest()`, `AutoRequest()`

## 1.11 intuition.library/BuildSysRequest

#### NAME

`BuildSysRequest` -- Build and display a system requester.

#### SYNOPSIS

```
ReqWindow = BuildSysRequest( Window, BodyText, PosText, NegText,
D0                          A0      A1      A2      A3
                          IDCMPFlags, Width, Height )
D0                          D2      D3
```

```
struct Window *BuildSysRequest( struct Window *, struct IntuiText *,
                                struct IntuiText *, struct IntuiText *, ULONG, WORD, WORD );
```

#### FUNCTION

This procedure builds a system requester based on the supplied information. If all goes well and the requester is constructed, this procedure returns a pointer to the window in which the requester appears. That window will have its IDCMP initialized to reflect the flags found in the IDCMPFlags argument. You may then wait on those ports to detect the user's response to your requester, which response may include either selecting one of the gadgets or causing some other event to be noticed by Intuition (like IDCMP\_DISKINSERTED, for instance). After the requester is satisfied, you should call the FreeSysRequest() procedure to remove the requester and free up any allocated memory.

See the autodoc for SysReqHandler() for more information on the how to handle the IntuiMessages this window will receive.

The requester used by this function has the NOISYREQ flag bit set, which means that the set of IDCMPFlags that may be used here include IDCMP\_RAWKEY, IDCMP\_MOUSEBUTTONS, and others.

In release previous to V36, if the requester could not be built, this function would try to call DisplayAlert() with the same information, with more or less favorable results. In V36, the requesters themselves require less memory (SIMPLEREQ), but there is no alert attempt.

The function may return TRUE (1) or FALSE if it cannot post the requester. (V36 will always return FALSE, but be sure to test for TRUE in case somebody reinstates the fallback alert.)

If the window argument you supply is equal to NULL, a new window will be created for you in the Workbench screen, or the default public screen, for V36. If you want the requester created by this routine to be bound to a particular window (i.e., to appear in the same screen as the window), you should not supply a window argument of NULL.

New for V36: if you pass a NULL window pointer, the system requester will appear on the default public screen, which is not always the Workbench.

The text arguments are used to construct the display. Each is a pointer to an instance of the structure IntuiText.

The BodyText argument should be used to describe the nature of the requester. As usual with IntuiText data, you may link several lines of text together, and the text may be placed in various locations in the requester. This IntuiText pointer will be stored in the ReqText variable of the new requester.

The PostText argument describes the text that you want associated with the user choice of "Yes, TRUE, Retry, Good." If the requester is successfully opened, this text will be rendered in a gadget in the lower-left of the requester, which gadget will have the

GadgetID field set to TRUE. If the requester cannot be opened and the DisplayAlert() mechanism is used, this text will be rendered in the lower-left corner of the alert display with additional text specifying that the left mouse button will select this choice. This pointer can be set to NULL, which specifies that there is no TRUE choice that can be made.

The NegText argument describes the text that you want associated with the user choice of "No, FALSE, Cancel, Bad." If the requester is successfully opened, this text will be rendered in a gadget in the lower-right of the requester, which gadget will have the GadgetID field set to FALSE. If the requester cannot be opened and the DisplayAlert() mechanism is used, this text will be rendered in the lower-right corner of the alert display with additional text specifying that the right mouse button will select this choice. This pointer cannot be set to NULL. There must always be a way for the user to cancel this requester.

The Positive and Negative Gadgets created by this routine have the following features:

- GTYP\_BOOLGADGET
- GACT\_RELVERIFY
- GTYP\_REQGADGET
- GACT\_TOGGLESELECT

When defining the text for your gadgets, you may find it convenient to use the special constants used by Intuition for the construction of the gadgets. These include defines like AUTODRAWMODE, AUTOLEFTEDGE, AUTOTOPEDGE and AUTOFRONTPEN. You can find these in your local intuition.h (or intuition.i) file.

These hard-coded constants are not very resolution or font sensitive, but V36 will override them to provide more modern layout.

New for V36, linked lists of IntuiText are not correctly supported for gadget labels.

The width and height values describe the size of the requester. All of your BodyText must fit within the width and height of your requester. The gadgets will be created to conform to your sizes.

VERY IMPORTANT NOTE: for this release of this procedure, a new window is opened in the same screen as the one containing your window. Future alternatives may be provided as a function distinct from this one.

NOTE: This function will pop the screen the requester and its window appears in to the front of all screens. New for V36, if the user doesn't perform any other screen arrangement before finishing with the requester, a popped screen will be pushed back behind.

#### INPUTS

Window = pointer to a Window structure  
BodyText = pointer to an IntuiText structure  
PosText = pointer to an IntuiText structure

---

NegText = pointer to an IntuiText structure  
 IDCMPFlags = the IDCMP flags you want used for the initialization of  
 the IDCMP of the window containing this requester  
 Width, Height = the size required to render your requester

NOTE for V36: the width and height you pass are ignored, as  
 are some of the parameters of your IntuiText, so that Intuition  
 can make the Requesters real nice for the new look.

#### RESULT

If the requester was successfully created, the value  
 returned by this procedure is a pointer to the window in which the  
 requester is rendered. If the requester could not be created,  
 this routine might have called DisplayAlert() before returning  
 (it depends on the version) and will pass back TRUE if the user  
 pressed the left mouse button and FALSE if the user pressed the  
 right mouse button. If the version of Intuition doesn't  
 call DisplayAlert(), or if it does, and there's not enough  
 memory for the alert, the value of FALSE is returned.

#### BUGS

This procedure currently opens a window in the Screen which  
 contains the window which is passed as a parameter, or the  
 default public screen, if that parameter is NULL. Although  
 not as originally envisioned, this will probably always be the  
 behavior of this function.

DisplayAlert() is not called in version V36.

It's almost impossible to make complete, correct account  
 of different system fonts, window border dimensions, and  
 screen resolution to get the layout of a System Requester  
 just right using this routine. For V36, we recommend the  
 automatic layout implemented in BuildEasyRequest and EasyRequest.

#### SEE ALSO

FreeSysRequest(), DisplayAlert(), ModifyIDCMP(), exec.library/Wait(),  
 Request(), AutoRequest(), EasyRequest(), BuildEasyRequestArgs()

## 1.12 intuition.library/ChangeWindowBox

#### NAME

ChangeWindowBox -- Change window position and dimensions. (V36)

#### SYNOPSIS

```
ChangeWindowBox( Window, Left, Top, Width, Height )
                A0      D0      D1      D2      D3
```

```
VOID ChangeWindowBox( struct Window *, WORD, WORD, WORD, WORD );
```

#### FUNCTION

Makes simultaneous changes in window position and dimensions,  
 in absolute (not relative) coordinates.

Like MoveWindow() and SizeWindow(), the effect of this function

is deferred until the next input comes along. Unlike these functions, `ChangeWindowBox()` specifies absolute window position and dimensions, not relative. This makes for more reliable results considering that the action is deferred, so this function is typically preferable to `MoveWindow()` and `SizeWindow()` paired.

You can detect that this operation has completed by receiving the `IDCMP_CHANGEWINDOW IDCMP` message

The dimensions are limited to legal range, but you should still take care to specify sensible inputs based on the window's dimension limits and the size of its screen.

This function limits the position and dimensions to legal values.

#### INPUTS

Window = the window to change position/dimension  
Left, Top, Width, Height = new position and dimensions

#### RESULT

Position and dimension are changed to your specification, or as close as possible.  
Returns nothing.

#### BUGS

#### SEE ALSO

`MoveWindow()`, `SizeWindow()`, `ZipWindow()`,  
`layers.library/MoveSizeLayer()`

## 1.13 intuition.library/ClearDMRequest

#### NAME

`ClearDMRequest` -- Clear (detaches) the `DMRequest` of the window.

#### SYNOPSIS

```
Response = ClearDMRequest( Window )
D0                                A0
```

```
BOOL ClearDMRequest( struct Window * );
```

#### FUNCTION

Attempts to clear the `DMRequest` from the specified window, that is detaches the special requester that you attach to the double-click of the menu button which the user can then bring up on demand. This routine WILL NOT clear the `DMRequest` if it's active (in use by the user). The `IDCMP` message class `IDCMP_REQCLEAR` can be used to detect that the requester is not in use, but that message is sent only when the last of perhaps several requesters in use in a window is terminated.

#### INPUTS

Window = pointer to the window from which the `DMRequest` is to be

---

cleared.

#### RESULT

If the DMRequest was not currently in use, detaches the DMRequest from the window and returns TRUE.

If the DMRequest was currently in use, doesn't change anything and returns FALSE.

#### BUGS

#### SEE ALSO

SetDMRequest(), Request()

## 1.14 intuition.library/ClearMenuStrip

#### NAME

ClearMenuStrip -- Clear (detach) the menu strip from the window.

#### SYNOPSIS

```
ClearMenuStrip( Window )
                A0
```

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

#### FUNCTION

Detaches the current menu strip from the window; menu strips are attached to windows using the SetMenuStrip() function (or, for V36, ResetMenuStrip() ).

If the menu is in use (for that matter if any menu is in use) this function will block (Wait()) until the user has finished.

Call this function before you make any changes to the data in a Menu or MenuItem structure which is part of a menu strip linked into a window.

#### INPUTS

Window = pointer to a window structure

#### RESULT

None

#### BUGS

#### SEE ALSO

SetMenuStrip(), ResetMenuStrip()

## 1.15 intuition.library/ClearPointer

#### NAME

ClearPointer -- Clear the mouse pointer definition from a window.

---

## SYNOPSIS

```
ClearPointer( Window )
            A0
```

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

## FUNCTION

Clears the window of its own definition of the Intuition mouse pointer. After calling ClearPointer(), every time this window is the active one the default Intuition pointer will be the pointer displayed to the user. If your window is the active one when this routine is called, the change will take place immediately.

Custom definitions of the mouse pointer which this function clears are installed by a call to SetPointer().

## INPUTS

Window = pointer to the window to be cleared of its pointer definition

## RESULT

None

## BUGS

## SEE ALSO

SetPointer()

## 1.16 intuition.library/CloseScreen

## NAME

CloseScreen -- Close an Intuition screen.

## SYNOPSIS

```
[Success =] CloseScreen( Screen )
            [D0]                A0
```

```
[BOOL] CloseScreen( struct Screen * );
/* returns BOOL in V36 and greater */
```

## FUNCTION

Unlinks the screen, unlinks the viewport, deallocates everything that Intuition allocated when the screen was opened (using OpenScreen()). Doesn't care whether or not there are still any windows attached to the screen. Doesn't try to close any attached windows; in fact, ignores them altogether (but see below for changes in V36).

If this is the last screen to go, attempts to reopen Workbench.

New for V36: this function will refuse to close the screen if there are windows open on the screen when CloseScreen() is called. This avoids the almost certain crash when a screen is closed out from under a window.

## INPUTS



Screen = pointer to the screen to be closed.

#### RESULT

New for V36: returns TRUE (1) if screen is closed,  
returns FALSE (0) if screen had open windows when  
called.

#### BUGS

#### SEE ALSO

OpenScreen()

## 1.17 intuition.library/CloseWindow

#### NAME

CloseWindow -- Close an Intuition window.

#### SYNOPSIS

```
CloseWindow( Window )  
            A0
```

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

#### FUNCTION

Closes an Intuition window. Unlinks it from the system, deallocates its memory, and makes it disappear.

When this function is called, all IDCMP messages which have been sent to your window are deallocated. If the window had shared a message Port with other windows, you must be sure that there are no unreplied messages for this window in the message queue. Otherwise, your program will try to make use of a linked list (the queue) which contains free memory (the old messages). This will give you big problems. See the code fragment CloseWindowSafely(), below.

NOTE: If you have added a Menu strip to this Window (via a call to SetMenuStrip()) you must be sure to remove that Menu strip (via a call to ClearMenuStrip()) before closing your Window.

NOTE: This function may block until it is safe to de-link and free your window. Your program may thus be suspended while the user plays with gadgets, menus, or window sizes and position.

New for V36: If your window is a "Visitor Window" (see OpenWindow) CloseWindow will decrement the "visitor count" in the public screen on which the window was open. When the last visitor window is closed, a signal will be sent to the public screen task, if this was pre-arranged (see OpenScreen).

#### INPUTS

Window = a pointer to a Window structure

#### RESULT

None

---

BUGS

SEE ALSO

OpenWindow(), OpenScreen(), CloseScreen()

EXAMPLE

```
/* CloseWindowSafely */

/* these functions close an Intuition window
 * that shares a port with other Intuition
 * windows or IPC customers.
 *
 * We are careful to set the UserPort to
 * null before closing, and to free
 * any messages that it might have been
 * sent.
 */
#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/ports.h"
#include "intuition/intuition.h"

CloseWindowSafely( win )
struct Window *win;
{
    /* we forbid here to keep out of race conditions with Intuition */
    Forbid();

    /* send back any messages for this window
     * that have not yet been processed
     */
    StripIntuiMessages( win->UserPort, win );

    /* clear UserPort so Intuition will not free it */
    win->UserPort = NULL;

    /* tell Intuition to stop sending more messages */
    ModifyIDCMP( win, 0L );

    /* turn multitasking back on */
    Permit();

    /* and really close the window */
    CloseWindow( win );
}

/* remove and reply all IntuiMessages on a port that
 * have been sent to a particular window
 * (note that we don't rely on the ln_Succ pointer
 * of a message after we have replied it)
 */
StripIntuiMessages( mp, win )
struct MsgPort *mp;
struct Window *win;
{
    struct IntuiMessage *msg;
```

---

```
struct Node *succ;

msg = (struct IntuiMessage *) mp->mp_MsgList.lh_Head;

while( succ = msg->ExecMessage.mn_Node.ln_Succ ) {

    if( msg->IDCMPWindow == win ) {

        /* Intuition is about to free this message.
         * Make sure that we have politely sent it back.
         */
        Remove( msg );

        ReplyMsg( msg );
    }

    msg = (struct IntuiMessage *) succ;
}
}
```

## 1.18 intuition.library/CloseWorkBench

### NAME

CloseWorkBench -- Closes the Workbench screen.

### SYNOPSIS

```
Success = CloseWorkBench()
D0
```

```
LONG CloseWorkBench( VOID );
```

### FUNCTION

This routine attempts to close the Workbench screen:

- Test whether or not any applications have opened windows on the Workbench, and return FALSE if so. Otherwise ...
- Clean up all special buffers
- Close the Workbench screen
- Make the Workbench program mostly inactive (it will still monitor disk activity)
- Return TRUE

### INPUTS

None

### RESULT

TRUE if the Workbench screen closed successfully  
FALSE if the Workbench was not open, or if it has windows open which are not Workbench drawers.

### NOTES

This routine has been drastically rewritten for V36.  
It is much more solid, although we haven't eliminated all the problem cases yet.

### BUGS

---

The name of this function is improperly spelled. Should be `CloseWorkbench()`.

It might be more convenient to have it return `TRUE` if the Workbench wasn't opened when called. The idea as it is now is probably this: if you want to free up the memory of the Workbench screen when your program begins, you can call `CloseWorkBench()`. The return value of that call indicates whether you should call `OpenWorkBench()` when your program exits: if `FALSE`, that means either the the Workbench existed but you could not close it, or that it wasn't around to begin with, and you should not try to re-open it.

We would prefer that you provide a user selection to attempt to open or close the Workbench screen from within your application, rather than your making assumptions like these.

SEE ALSO

`OpenWorkBench()`

## 1.19 intuition.library/CurrentTime

NAME

`CurrentTime -- Get the current time values.`

SYNOPSIS

```
CurrentTime( Seconds, Micros )
              A0          A1
```

```
VOID CurrentTime( ULONG *, ULONG * );
```

FUNCTION

Puts copies of the current time into the supplied argument pointers.

This time value is not extremely accurate, nor is it of a very fine resolution. This time will be updated no more than sixty times a second, and will typically be updated far fewer times a second.

INPUTS

Seconds = pointer to a `LONG` variable to receive the current seconds value

Micros = pointer to a `LONG` variable for the current microseconds value

RESULT

Puts the time values into the memory locations specified by the arguments

Return value is not defined.

BUGS

SEE ALSO

`timer.device/TR_GETSYSTIME`

---

## 1.20 intuition.library/DisplayAlert

### NAME

DisplayAlert -- Create the display of an alert message.

### SYNOPSIS

```
Response = DisplayAlert( AlertNumber, String, Height )
D0                      D0                      A0      D1
```

```
BOOL DisplayAlert( ULONG, UBYTE *, WORD );
```

### FUNCTION

Creates an alert display with the specified message.

If the system can recover from this alert, it's a RECOVERY\_ALERT and this routine waits until the user presses one of the mouse buttons, after which the display is restored to its original state and a BOOL value is returned by this routine to specify whether or not the user pressed the LEFT mouse button.

If the system cannot recover from this alert, it's a DEADEND\_ALERT and this routine returns immediately upon creating the alert display. The return value is FALSE.

NOTE: Starting with V33, if Intuition can't get enough memory for a RECOVERY\_ALERT, the value FALSE will be returned.

AlertNumber is a LONG value, historically related to the value sent to the Alert() routine. But the only bits that are pertinent to this routine are the ALERT\_TYPE bit(s). These bits must be set to either RECOVERY\_ALERT for alerts from which the system may safely recover, or DEADEND\_ALERT for those fatal alerts. These states are described in the paragraph above. There is a third type of alert, the DAISY\_ALERT, which is used only by the Exec.

The string argument points to an AlertMessage string. The AlertMessage string is comprised of one or more substrings, each of which is composed of the following components:

- first, a 16-bit x-coordinate and an 8-bit y-coordinate, describing where on the alert display you want this string to appear. The y-coordinate describes the offset to the baseline of the text.
- then, the bytes of the string itself, which must be null-terminated (end with a byte of zero)
- lastly, the continuation byte, which specifies whether or not there's another substring following this one. If the continuation byte is non-zero, there IS another substring to be processed in this alert message. If the continuation byte is zero, this is the last substring in the message.

The last argument, Height, describes how many video lines tall you want the alert display to be.

New for V36: Alerts are always rendered in Topaz 8 (80 column font), regardless of the system default font. Also, RECOVERY\_ALERTs are displayed in amber, while DEADEND\_ALERTs are still red. Alerts

---

no longer push down the application screens to be displayed. Rather, they appear alone in a black display.

Also new for V36: Alerts block each other out, and input during an alert is deprived of the rest of the system. Internal input buffers still cause alert clicks to be processed by applications sometimes.

#### INPUTS

AlertNumber = the number of this alert message. The only pertinent bits of this number are the ALERT\_TYPE bit(s). The rest of the number is ignored by this routine.

String = pointer to the alert message string, as described above

Height = minimum display lines required for your message

#### RESULT

A BOOL value of TRUE or FALSE. If this is a DEADEND\_ALERT, FALSE is always the return value. If this is a RECOVERY\_ALERT. The return value will be TRUE if the user presses the left mouse button in response to your message, and FALSE if the user presses the right hand button in response to your text, or if the alert could not be posted.

#### BUGS

If the system is worse off than you think, the level of your alert may become DEADEND\_ALERT without you ever knowing about it. This will NOT happen due simply to low memory. Rather, the alert display will be skipped, and FALSE will be returned.

The left and right button clicks satisfying the alerts are unfortunately passed to the normal applications, because of some internal system input buffering.

#### SEE ALSO

## 1.21 intuition.library/DisplayBeep

#### NAME

DisplayBeep -- Flash the video display.

#### SYNOPSIS

```
DisplayBeep( Screen )
            A0
```

```
VOID DisplayBeep( struct Screen * );
```

#### FUNCTION

"Beeps" the video display by flashing the background color of the specified screen. If the screen argument is NULL, every screen in the display will be beeped. Flashing everyone's screen is not a polite thing to do, so this should be reserved for dire circumstances.

The reason such a routine is supported is because the Amiga has no internal bell or speaker. When the user needs to know of

---

an event that is not serious enough to require the use of a requester, the `DisplayBeep()` function may be called.

New for V36: Intuition now calls `DisplayBeep` through the external library vector. This means that if you call `SetFunction()` to replace `DisplayBeep` with an audible beep, for example, then your change will affect even Intuition's calls to `DisplayBeep`.

#### INPUTS

Screen = pointer to a screen. If NULL, every screen in the display will be flashed

#### RESULT

None

#### BUGS

#### SEE ALSO

## 1.22 intuition.library/DisposeObject

#### NAME

`DisposeObject` -- Deletes a 'boopsi' object. (V36)

#### SYNOPSIS

```
DisposeObject( Object )
              A0
```

```
VOID DisposeObject( APTR );
```

#### FUNCTION

Deletes a boopsi object and all of its auxiliary data. These objects are all created by `NewObject()`. Objects of certain classes "own" other objects, which will also be deleted when the object is passed to `DisposeObject()`. Read the per-class documentation carefully to be aware of these instances.

#### INPUTS

Object = abstract pointer to a boopsi object returned by `NewObject()`

#### NOTES

This function invokes the `OM_DISPOSE` method.

#### RESULT

None.

#### BUGS

#### SEE ALSO

`NewObject()`, `SetAttrs()`, `GetAttr()`, `MakeClass()`,  
Document "Basic Object-Oriented Programming System for Intuition"  
and the "boopsi Class Reference" document.

---

## 1.23 intuition.library/DoubleClick

### NAME

DoubleClick -- Test two time values for double-click timing.

### SYNOPSIS

```
IsDouble = DoubleClick( StartSecs, StartMicros,
                        D0          D0          D1
                        CurrentSecs, CurrentMicros )
                        D2          D3
```

```
BOOL DoubleClick( ULONG, ULONG, ULONG, ULONG );
```

### FUNCTION

Compares the difference in the time values with the double-click timeout range that the user has set (using the "Preferences" tool) or some other program has configured into the system. If the difference between the specified time values is within the current double-click time range, this function returns TRUE, else it returns FALSE.

These time values can be found in input events and IDCMP messages. The time values are not perfect; however, they are precise enough for nearly all applications.

### INPUTS

StartSeconds, StartMicros = the timestamp value describing the start of the double-click time period you are considering  
CurrentSeconds, CurrentMicros = the timestamp value describing the end of the double-click time period you are considering

### RESULT

If the difference between the supplied timestamp values is within the double-click time range in the current set of Preferences, this function returns TRUE, else it returns FALSE

### BUGS

### SEE ALSO

CurrentTime()

## 1.24 intuition.library/DrawBorder

### NAME

DrawBorder -- Draw the specified Border structure into a RastPort.

### SYNOPSIS

```
DrawBorder( RastPort, Border, LeftOffset, TopOffset )
           A0          A1          D0          D1
```

```
VOID DrawBorder( struct RastPort *, struct Border *, WORD, WORD );
```

### FUNCTION

First, sets up the draw mode and pens in the RastPort according to the

---



arguments of the Border structure. Then, draws the vectors of the border argument into the RastPort, offset by the left and top offsets.

As with all graphics rendering routines, the border will be clipped to the boundaries of the RastPort's layer, if it exists. This is the case with window RastPorts.

This routine will draw all borders in the NULL-terminated list linked by the NextBorder field of the border argument.

#### INPUTS

RastPort = pointer to the RastPort to receive the border rendering  
 Border = pointer to a Border structure  
 LeftOffset = the offset to be added to each vector's x coordinate  
 TopOffset = the offset to be added to each vector's y coordinate

#### RESULT

None

#### BUGS

#### SEE ALSO

## 1.25 intuition.library/DrawImage

#### NAME

DrawImage -- Draw the specified Image structure into a RastPort.

#### SYNOPSIS

```
DrawImage( RastPort, Image, LeftOffset, TopOffset )
           A0          A1          D0          D1
```

```
VOID DrawImage( struct RastPort *, struct Image *, WORD, WORD );
```

#### FUNCTION

First, sets up the draw mode and pens in the RastPort according to the arguments of the Image structure. Then, moves the image data of the image argument into the RastPort, offset by the left and top offsets.

This routine does window layer clipping if you pass your window's (layered) RastPort -- if you draw an image outside of your window, your imagery will be clipped at the window's edge. If you pass a (non-layered) screen RastPort, you MUST be sure your image is wholly contained within the rastport bounds.

If the NextImage field of the image argument is non-NULL, the next image is rendered as well, and so on until some NextImage field is found to be NULL.

#### INPUTS

RastPort = pointer to the RastPort to receive image rendering  
 Image = pointer to an image structure  
 LeftOffset = the offset which will be added to the image's x coordinate

---

TopOffset = the offset which will be added to the image's y coordinate

#### RESULT

None

#### NOTES

Intuition always has and will continue to assume there are at least as many planes of data pointed to by ImageData as there are '1' bits in the PlanePick field. Please ensure that this is so. (See the intuition.h include file for full details on using PlanePick).

#### BUGS

#### SEE ALSO

DrawImageState(), EraseImage()

## 1.26 intuition.library/DrawImageState

#### NAME

DrawImageState -- Draw an (extended) Intuition Image with special visual state. (V36)

#### SYNOPSIS

```
DrawImageState( RPort, Image, LeftOffset, TopOffset, State, DrawInfo )
                A0      A1      D0          D1          D2      A2
```

```
VOID DrawImageState( struct RastPort *, struct Image *,
                    WORD, WORD, ULONG, struct DrawInfo * );
```

#### FUNCTION

This function draws an Intuition Image structure in a variety of "visual states," which are defined by constants in intuition/imageclass.h. These include:

- IDS\_NORMAL               - like DrawImage()
- IDS\_SELECTED            - represents the "selected state" of a Gadget
- IDS\_DISABLED            - the "ghosted state" of a gadget
- IDS\_BUSY                - for future functionality
- IDS\_INDETERMINATE       - for future functionality
- IDS\_INACTIVENORMAL      - for gadgets in window border
- IDS\_INACTIVESELECTED    - for gadgets in window border
- IDS\_INACTIVEDISABLED    - for gadgets in window border

Only IDS\_NORMAL will make sense for traditional Image structures, this function is more useful when applied to new custom images or "object-oriented image classes."

Each class of custom images is responsible for documenting which visual states it supports, and you typically want to use images which support the appropriate states with your custom gadgets.

The DrawInfo parameter provides information invaluable to "rendered" images, such as pen color and resolution. Each image class must document whether this parameter is required.

## INPUTS

RPort - RastPort for rendering  
 Image - pointer to a (preferably custom) image  
 LeftOffset,RightOffset - positional offsets in pixels  
 State - visual state selected from above  
 DrawInfo - pointer to packed of pen selections and resolution.

## RESULT

None.

## EXAMPLE

Provided separately in the DevCon '90 disk set.

## NOTES

## BUGS

## SEE ALSO

DrawImage(), GetScreenDrawInfo(), intuition/imageclass.h

## 1.27 intuition.library/EasyRequestArgs

## NAME

EasyRequestArgs -- Easy alternative to AutoRequest(). (V36)  
 EasyRequest -- Varargs stub for EasyRequestArgs(). (V36)

## SYNOPSIS

```
num = EasyRequestArgs( Window, easyStruct, IDCMP_ptr, ArgList )
D0                                A0      A1      A2      A3
```

```
LONG EasyRequestArgs( struct Window *, struct EasyStruct *,
                     ULONG *, APTR );
```

```
num = EasyRequest( Window, easyStruct, IDCMP_ptr, Arg1, Arg2, ... )
```

```
LONG EasyRequest( struct Window *, struct EasyStruct *,
                 ULONG *, APTR, ... );
```

```
( from intuition.h )
struct EasyStruct {
    ULONG      es_StructSize;
    ULONG      es_Flags;
    UBYTE      *es_Title;
    UBYTE      *es_TextFormat;
    UBYTE      *es_GadgetFormat;
};
```

## FUNCTION

This function provides a simpler method of using a 'System Requester' than provided by AutoRequest(). It performs layout and size calculations sensitive to the current font and screen resolution.

It provides for the descriptive 'body' text and the gadget text to be constructed from 'printf' style format strings.

It also provides a general way for the requester to be sensitive to particular IDCMP messages.

The first function listed is the actual Intuition library function. It is passed the arguments for the formatting operations as a pointer to the first argument.

The second function uses a C-style variable number of argument (varargs) calling convention. It should be implemented as a call to the first function, and might be supplied by your compiler vendor, in `amiga.lib`, or using the first example below, for most C compilers.

NOTE: The formatting is done by `exec.library/RawDoFmt()`, so be aware that to display a 32-bit integer argument, for example, you must say `"%ld"`, not `"%d"`, since `RawDoFmt()` is "word-oriented."

NOTE: This function switches the processor stack to ensure sufficient stack space for the function to complete.

#### EXAMPLES

```
/* varargs interface works for most C compilers */
EasyRequest( w, es, ip, arg1 )
struct Window      *w;
struct EasyStruct  *es;
ULONG              *ip;
int                arg1;
{
    return ( EasyRequestArgs( w, es, ip, &arg1 ) );
}

/*****

/* typical use */
struct EasyStruct volumeES = {
    sizeof (struct EasyStruct),
    0,
    "Volume Request",
    "Please insert volume %s in any drive.",
    "Retry|Cancel",
};
#define CANCEL    (0)

Volume *
getVolume( volname )
UBYTE  *volname;
{
    Volume      *vptr;
    Volume      *findVolume();
    UWORD       reply;
    ULONG       iflags;

    iflags = IDCMP_DISKINSERTED;

    while ( ((vptr = findVolume( volname )) == NULL) &&
```

---

```

(EasyRequest( w, &volumeES, &iflags, volname ) != CANCEL) )
    /* loop */ ;

/* note that in some circumstances, you will have to
   re-initialize the value of 'iflags'. Here, it
   is either unchanged, or returned as the single
   IDCMPFlag value IDCMP_DISKINSERTED. If you combine
   multiple IDCMPFlag values in 'iflags,' only
   one will be returned, so you must reinitialize
   'iflags' to be the combination.
*/
return ( vptr );
}

```

#### INPUTS

Window = Reference window pointer, determines the screen and title of the requester window. This can be NULL, which means the requester is to appear on the Workbench screen, or default public screen, if defined.

IDCMP\_ptr = Pointer to IDCMP flags that you want to terminate the requester. This pointer may be NULL.

easyStruct = Pointer to EasyStruct structure with fields interpreted as follows:

es\_StructSize = sizeof (struct EasyStruct), for future extension.

es\_Flags = 0 for now, in the future may specify other options.

es\_Title = Title of system requester window. If this is NULL, the title will be taken to be the same as the title of 'Window', if provided, or else "System Request."

es\_TextFormat = Format string, a la RawDoFmt(), for message in requester body. Lines are separated by '\n'. Formatting '%' functions are supported exactly as in RawDoFmt().

es\_GadgetFormat = Format string for gadgets. Text for separate gadgets is separated by '|'. Format functions are supported. You MUST specify at least one gadget.

Args = Arguments for format commands. Arguments for GadFmt follow arguments for TextFmt.

#### RESULT

0, 1, ..., N = Successive GadgetID values, for the gadgets you specify for the requester. NOTE: The numbering from left to right is actually: 1, 2, ..., N, 0. This is for compatibility with AutoRequests which has FALSE for the rightmost gadget.

-1 = Means that one of the caller-supplied IDCMPFlags occurred. The IDCMPFlag value is in the longword pointed to by IDCMP\_ptr.

#### BUGS

Does not fall back to a recoverable alert if the requester cannot be created.

Does not handle case when gadgets don't fit or window title is too long, although it does trim trailing spaces from the title for calculating dimensions.

## PLANS

Possible enhancements include: centering of text, size-sensitive layout, window-relative requester, vertical gadget layout, window placement, more keyboard shortcuts.

We also reserve the use of the newline character '\n' in gadget format strings for future use as a line separator.

## SEE ALSO

`exec.library/RawDoFmt()`, `BuildEasyRequestArgs()`, `SysReqHandler()`, `AutoRequest()`, `BuildSysRequest()`

## 1.28 intuition.library/EndRefresh

## NAME

`EndRefresh` -- End the optimized refresh state of the window.

## SYNOPSIS

```
EndRefresh( Window, Complete )
           A0      D0
```

```
VOID EndRefresh( struct Window *, BOOL );
```

## FUNCTION

This function gets you out of the special refresh state of your window. It is called following a call to `BeginRefresh()`, which routine puts you into the special refresh state. While your window is in the refresh state, the only rendering that will be wrought in your window will be to those areas which were recently revealed and need to be refreshed.

After you've done all the refreshing you want to do for this window, you should call this routine to restore the window to its non-refreshing state. Then all rendering will go to the entire window, as usual.

The 'Complete' argument is a boolean TRUE or FALSE value used to describe whether or not the refreshing you've done was all the refreshing that needs to be done at this time. Most often, this argument will be TRUE. But if, for instance, you have multiple tasks or multiple procedure calls which must run to completely refresh the window, then each can call its own `Begin/EndRefresh()` pair with a Complete argument of FALSE, and only the last calls with a Complete argument of TRUE.

**WARNING:** Passing this function the value of FALSE has its pitfalls. Please see the several caveats in the autodoc for `BeginRefresh()`.

For your information, this routine calls the Layers library function `EndUpdate()`, unlocks your layers (calls `UnlockLayerRom()`), clears the `LAYERREFRESH` bit in your Layer Flags, and clears the `WFLG_WINDOWREFRESH` bit in your window Flags.

## INPUTS

Window = pointer to the window currently in optimized-refresh mode  
Complete = Boolean TRUE or FALSE describing whether or not this  
window is completely refreshed

RESULT  
None

BUGS

SEE ALSO  
BeginRefresh(), layers.library/EndUpdate(),  
graphics.library/UnlockLayerRom()

## 1.29 intuition.library/EndRequest

NAME  
EndRequest -- Remove a currently active requester.

SYNOPSIS  
EndRequest( Requester, Window )  
            A0            A1

VOID EndRequest( struct Requester \*, struct Window \* );

FUNCTION  
Ends the request by erasing the requester and decoupling it from  
the window.

Note that this doesn't necessarily clear all requesters from the  
window, only the specified one. If the window labors under other  
requesters, they will remain in the window.

INPUTS  
Requester = pointer to the requester to be removed  
Window = pointer to the Window structure with which this requester  
is associated

RESULT  
None

BUGS

SEE ALSO  
Request()

## 1.30 intuition.library/EraseImage

NAME  
EraseImage -- Erases an Image. (V36)

SYNOPSIS  
EraseImage( RPort, Image, LeftOffset, TopOffset )

---

A0      A1      D0      D1

```
VOID EraseImage( struct RastPort *, struct Image *, WORD, WORD );
```

#### FUNCTION

Erases an Image. For a normal Image structure, this will call the graphics function `EraseRect()` (clear using layer backfill, if any) for the Image box (LeftEdge/TopEdge/Width/Height).

For custom image, the exact behavior is determined by the custom image class.

#### INPUTS

RPort      - RastPort to erase a part of  
Image      - custom or standard image  
LeftOffset,RightOffset - pixel offsets of Image position

#### RESULT

None.

#### EXAMPLE

#### NOTES

#### BUGS

#### SEE ALSO

`graphics.library/EraseRect()`.

## 1.31 intuition.library/FreeClass

#### NAME

`FreeClass` -- Frees a boopsi class created by `MakeClass()`. (V36)

#### SYNOPSIS

```
success = FreeClass( ClassPtr )
D0                      A0
```

```
BOOL FreeClass( struct IClass * );
```

#### FUNCTION

For class implementors only.

Tries to free a boopsi class created by `MakeClass()`. This won't always succeed: classes with outstanding objects or with subclasses cannot be freed. You cannot allow the code which implements the class to be unloaded in this case.

For public classes, this function will *always* remove the class (see `RemoveClass()`) making it unavailable, whether it succeeds or not.

If you have a dynamically allocated data for your class (hanging off of `cl_UserData`), try to free the class before you free the user data, so you don't get stuck with a half-freed class.



## INPUTS

ClassPtr - pointer to a class created by MakeClass().

## RESULT

Returns FALSE if the class could not be freed. Reasons include, but will not be limited to, having non-zero cl\_ObjectCount or cl\_SubclassCount.

Returns TRUE if the class could be freed.

Calls RemoveClass() for the class in either case.

## EXAMPLE

Freeing a private class with dynamically allocated user data:

```
freeMyClass( cl )
struct IClass  *cl;
{
    struct MyPerClassData      *mpcd;

    mpcd = (struct MyPerClassData *) cl->cl_UserData;
    if ( FreeClass( cl ) )
    {
        FreeMem( mpcd, sizeof mpcd );
        return ( TRUE );
    }
    else
    {
        return ( FALSE );
    }
}
```

## BUGS

## SEE ALSO

MakeClass(),  
Document "Basic Object-Oriented Programming System for Intuition"  
and the "boopsi Class Reference" document.

## 1.32 intuition.library/FreeRemember

## NAME

FreeRemember -- Free memory allocated by calls to AllocRemember().

## SYNOPSIS

```
FreeRemember( RememberKey, ReallyForget )
                A0                D0
```

```
VOID FreeRemember( struct Remember **, BOOL );
```

## FUNCTION

This function frees up memory allocated by the AllocRemember() function. It will either free up just the Remember structures, which supply the link nodes that tie your allocations together, or it

will deallocate both the link nodes AND your memory buffers too.

If you want to deallocate just the Remember structure link nodes, you should set the ReallyForget argument to FALSE. However, if you want FreeRemember to really deallocate all the memory, including both the Remember structure link nodes and the buffers you requested via earlier calls to AllocRemember(), then you should set the ReallyForget argument to TRUE.

NOTE WELL: Once you call this function passing it FALSE, the linkages between all the memory chunks are lost, and you cannot subsequently use FreeRemember() to free them.

#### INPUTS

RememberKey = the address of a pointer to struct Remember. This pointer should either be NULL or set to some value (possibly NULL) by a call to AllocRemember().  
ReallyForget = a BOOL FALSE or TRUE describing, respectively, whether you want to free up only the Remember nodes or if you want this procedure to really forget about all of the memory, including both the nodes and the memory buffers referenced by the nodes.

#### EXAMPLE

```
struct Remember *RememberKey;
RememberKey = NULL;
AllocRemember(&RememberKey, BUFSIZE, MEMF_CHIP);
FreeRemember(&RememberKey, TRUE);
```

#### RESULT

None

#### BUGS

#### SEE ALSO

AllocRemember(), exec.library/FreeMem()

## 1.33 intuition.library/FreeScreenDrawInfo

#### NAME

FreeScreenDrawInfo -- Finish using a DrawInfo structure. (V36)

#### SYNOPSIS

```
FreeScreenDrawInfo( Screen, DrInfo )
                   A0      A1
```

```
VOID FreeScreenDrawInfo( struct Screen *, struct DrawInfo * );
```

#### FUNCTION

Declares that you are finished with the DrawInfo structure returned by GetScreenDrawInfo().

#### INPUTS

Screen                    - pointer to screen passed to GetScreenDrawInfo()  
DrInfo                    - pointer to DrawInfo returned by GetScreenDrawInfo()

## RESULT

None

## NOTES

This function, and `GetScreenDrawInfo()`, don't really do much, but they provide an upward compatibility path. That means that if you misuse them today, they probably won't cause a problem, although they may someday later. So, please be very careful only to use the `DrawInfo` structure between calls to `GetScreenDrawInfo()` and `FreeScreenDrawInfo()`, and be sure that you don't forget `FreeScreenDrawInfo()`.

## BUGS

## SEE ALSO

`GetScreenDrawInfo()`

## 1.34 intuition.library/FreeSysRequest

## NAME

`FreeSysRequest` -- Free resources gotten by a call to `BuildSysRequest()`.

## SYNOPSIS

```
FreeSysRequest( Window )
               A0
```

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

## FUNCTION

This routine frees up all memory allocated by a successful call to the `BuildSysRequest()` procedure. If `BuildSysRequest()` returned a pointer to a window, then you are able to wait on the message port of that window to detect an event which satisfies the requester. When you want to remove the requester, you call this procedure. It ends the requester and deallocates any memory used in the creation of the requester. It also closes the special window that was opened for your system requester.

For V36: It's OK if you pass a `NULL` or a `TRUE` (1) value to this function. Also, this function properly disposes of requesters gotten using `BuildEasyRequest()`.

## INPUTS

`Window` = value of the window pointer returned by a successful call to the `BuildSysRequest()` procedure

## RESULT

None

## BUGS

## SEE ALSO

`BuildSysRequest()`, `AutoRequest()`, `CloseWindow()`

---

## 1.35 intuition.library/GadgetMouse

### NAME

GadgetMouse -- Calculate gadget-relative mouse position. (V36)

### SYNOPSIS

```
GadgetMouse( Gadget, GInfo, MousePoint )
              A0      A1      A2
```

```
VOID GadgetMouse( struct GadgetInfo *, WORD * );
```

### FUNCTION

Determines the current location of the mouse pointer relative to the upper-left corner of a custom gadget. Typically used only in the GM\_HANDLEINPUT and GM\_GOACTIVE custom gadget hook routines.

NEWS FLASH!!: These two hook routines are now passed the mouse coordinates, so this function has no known usefulness.

We recommend that you don't call it.

Note that this function calculates the mouse position taking "gadget relativity" (GFLG\_RELRIGHT, GFLG\_RELBOTTOM) into consideration. If your custom gadget intends to ignore these properties, then you should either enjoin or inhibit your users from setting those bits, since Intuition won't ask if you respect them.

### INPUTS

GInfo = A pointer to a GadgetInfo structure as passed to the custom gadget hook routine.

MousePoint = address of two WORDS, or a pointer to a structure of type Point.

### RESULT

Returns nothing. Fills in the two words pointed to by MousePoint with the gadget-relative mouse position.

### BUGS

Useless, since equivalent information is now passed to every function that might have a use for this.

### SEE ALSO

## 1.36 intuition.library/GetAttr

### NAME

GetAttr -- Inquire the value of some attribute of an object. (V36)

### SYNOPSIS

```
attr = GetAttr( AttrID, Object, StoragePtr )
              D0      A0      A1
```

```
ULONG GetAttr( ULONG, APTR, ULONG * );
```

#### FUNCTION

Inquires from the specified object the value of the specified attribute.

You always pass the address of a long variable, which will receive the same value that would be passed to SetAttrs() in the ti\_Data portion of a TagItem element. See the documentation for the class for exceptions to this general rule.

Not all attributes will respond to this function. Those that will are documented on a class-by-class basis.

#### INPUTS

AttrID = the attribute tag ID understood by the object's class  
Object = abstract pointer to the boopsi object you are interested in  
StoragePtr = pointer to appropriate storage for the answer

#### RESULT

Returns FALSE (0) if the inquiries of attribute are not provided by the object's class.

#### NOTES

This function invokes the OM\_GET method of the object.

#### BUGS

#### SEE ALSO

NewObject(), DisposeObject(), SetAttrs(), MakeClass(),  
Document "Basic Object-Oriented Programming System for Intuition"  
and the "boopsi Class Reference" document.

## 1.37 intuition.library/GetDefaultPubScreen

#### NAME

GetDefaultPubScreen -- Get name of default public screen. (V36)

#### SYNOPSIS

```
GetDefaultPubScreen( Namebuff )
                    A0
```

```
VOID GetDefaultPubScreen( UBYTE * );
```

#### FUNCTION

Provides the name of the current default public screen.  
Only anticipated use is for Public Screen Manager utilities,  
since it is easy to open a visitor window on the default  
public screen without specifying the name.

#### INPUTS

Namebuff = a buffer of MAXPUBSCREENNAME. This can be NULL.

#### RESULT

None. Will provide the string "Workbench" in Namebuff if there

---

is no current default public screen.

#### NOTE

This function actually "returns" in register D0 a pointer to the public screen. Unfortunately, the lifespan of this pointer is not ensured; the screen could be closed at any time. The \*ONLY\* legitimate use we can see for this return value is to compare for identity with the pointer to a public screen you either have a window open in, or a lock on using LockPubScreen(), to determine if that screen is in fact the default screen.

#### BUGS

The function prototype does not reflect the return value.

#### SEE ALSO

SetDefaultPubScreen(), OpenWindow()

## 1.38 intuition.library/GetDefPrefs

#### NAME

GetDefPrefs -- Get a copy of the the Intuition default Preferences.

#### SYNOPSIS

```
Prefs = GetDefPrefs( PrefBuffer, Size )  
D0                A0                D0
```

```
struct Preferences *GetDefPrefs( struct Preferences *, WORD );
```

#### FUNCTION

Gets a copy of the Intuition default preferences data. Writes the data into the buffer you specify. The number of bytes you want copied is specified by the size argument.

The default preferences are those that Intuition uses when it is first opened. If no preferences file is found, these are the preferences that are used. These would also be the startup preferences in an AmigaDOS-less environment.

It is legal to take a partial copy of the Preferences structure. The more pertinent preferences variables have been grouped near the top of the structure to facilitate the memory conservation that can be had by taking a copy of only some of the Preferences structure.

#### INPUTS

PrefBuffer = pointer to the memory buffer to receive your copy of the Intuition Preferences structure  
Size = the number of bytes in your PrefBuffer, the number of bytes you want copied from the system's internal Preference settings

#### RESULT

Returns your parameter PrefBuffer.

#### BUGS

---

SEE ALSO  
GetPrefs()

## 1.39 intuition.library/GetPrefs

### NAME

GetPrefs -- Get the current Intuition Preferences structure.

### SYNOPSIS

```
Prefs = GetPrefs( PrefBuffer, Size )  
D0                      A0          D0
```

```
struct Preferences *GetPrefs( struct Preferences *, WORD );
```

### FUNCTION

Gets a copy of the current Intuition Preferences structure.  
Writes the data into the buffer you specify. The number of bytes you want copied is specified by the size argument.

It is legal to take a partial copy of the Preferences structure. The more pertinent preferences variables have been grouped near the top of the structure to facilitate the memory conservation that can be had by taking a copy of only some of the Preferences structure.

New for V36: A new and more extensible method for supplying Preferences has been introduced in V36, and relies on file system notification. The Intuition preferences items rely also on the IPrefs program. Certain elements of the Preferences structure have been superceded by this new method. As much as possible, the Preferences structure returned by GetPrefs() reflect the current state of Preferences. However, it is impossible to represent some of the V36-style preferences items using the existing Preferences structure.

### INPUTS

PrefBuffer = pointer to the memory buffer to receive your copy of the Intuition Preferences  
Size = the number of bytes in your PrefBuffer, the number of bytes you want copied from the system's internal Preference settings

### RESULT

Returns your parameter PrefBuffer.

### BUGS

SEE ALSO  
GetDefPrefs(), SetPrefs()

## 1.40 intuition.library/GetScreenData

---

## NAME

GetScreenData -- Get copy of a screen data structure.

## SYNOPSIS

```
Success = GetScreenData( Buffer, Size, Type, Screen )
```

```
D0                                A0          D0      D1      A1
```

```
BOOL GetScreenData( APTR, UWORD, UWORD, struct Screen * );
```

## FUNCTION

This function copies into the caller's buffer data from a Screen structure. Typically, this call will be used to find the size, title bar height, and other values for a standard screen, such as the Workbench screen.

To get the data for the Workbench screen, one would call:

```
GetScreenData(buff, sizeof(struct Screen), WBENCHSCREEN, NULL)
```

NOTE: if the requested standard screen is not open, this function will have the effect of opening it.

This function has been useful for two basic types of things:

- 1) Determining information about the Workbench screen, in preparation for opening a window on it.
- 2) Attempts at discerning the user's preferences in a working screen, for "cloning" the Workbench modes and dimensions when opening a similar custom screen.

Providing compatibility with both of these goals has proven difficult, as we introduce new display modes and screen scrolling in V36. Read carefully the somewhat involved exceptions we elected to implement ...

Changes as of V36:

For V36 and later, the function LockPubScreen() is an improvement over this function, in that it doesn't copy the screen data but returns a pointer and a guarantee that the screen will not be closed.

If the global public screen SHANGHAI mode is in effect (see SetPubScreenModes() ), this function will actually report on the default public screen, where "Workbench" windows will actually open.

For V36 and later, this function does some "compatibility tricks" when you inquire about the WBENCHSCREEN. To keep programs from "stumbling" into modes they don't understand, and because an NTSC machine may be running a PAL Workbench or PRODUCTIVITY, for example, the following "false" information is returned.

The Screen.ViewPort.Modes field will either be HIRES or HIRES+LACE (with the SPRITES flag also set, as usual). HIRES+LACE is used if the display mode selected for the Workbench screen is an interlaced screen of any type.



The dimensions returned will be the \*smaller\* of the OSCAN\_TEXT dimensions for the returned mode, and the actual dimensions of the Workbench screen.

EXCEPTION: For specific compatibility considerations, if the Workbench is in one of the A2024 modes, the mode returned in Screen.ViewPort.Modes will be HIRES+LACE (with perhaps some "special" bits also set for future improvement), but with dimensions equal to the actual A2024-mode Workbench screen. This will favor programs which open windows on the A2024 Workbench, but will cause some problems for programs which try to "clone" the Workbench screen using this function.

If you want the real information about the modern Workbench screen, call LockPubScreen( "Workbench" ) and acquire its display mode ID by inquiring of the actual ViewPort (using graphics.library/GetVPMODEID() ).

You may then use the information you get to clone as many of the properties of the Workbench screen that you wish.

In the long run, it's probably better to provide your user with a screen mode selection option, and skip all this.

#### INPUTS

Buffer = pointer to a buffer into which data can be copied  
 Size = the size of the buffer provided, in bytes  
 Type = the screen type, as specified in OpenWindow() (WBENCHSCREEN, CUSTOMSCREEN, ...)  
 Screen = ignored, unless type is CUSTOMSCREEN, which results only in copying 'size' bytes from 'screen' to 'buffer'

#### RESULT

TRUE if successful  
 FALSE if standard screen of Type 'type' could not be opened.

#### BUGS

You cannot support the new V36 display modes using this function.

#### SEE ALSO

OpenWindow(), LockPubScreen(), graphics.library/GetVPMODEID(), SetPubScreenModes(), OpenScreen()

## 1.41 intuition.library/GetScreenDrawInfo

#### NAME

GetScreenDrawInfo -- Get pointer to rendering information. (V36)

#### SYNOPSIS

```
DrInfo = GetScreenDrawInfo( Screen )
D0                                     A0
```

```
struct DrawInfo *GetScreenDrawInfo( struct Screen * );
```

#### FUNCTION

---

Returns a pointer to a DrawInfo structure derived from the screen passed. This data structure is READ ONLY. The field dri\_Version identifies which version of struct DrawInfo you are given a pointer to.

#### INPUTS

Screen - pointer to a valid, open screen.

#### RESULT

DrInfo - pointer to a system-allocated DrawInfo structure, as defined in intuition/screens.h.

#### NOTES

Some information in the DrawInfo structure may in the future be calculated the first time this function is called for a particular screen.

You must call FreeScreenDrawInfo() when you are done using the returned pointer.

This function does not prevent a screen from closing. Apply it only to the screens you opened yourself, or apply a protocol such as LockPubScreen().

**WARNING:** Until further notice, the pointer returned does not remain valid after the screen is closed.

This function and FreeScreenDrawInfo() don't really do much now, but they provide an upward compatibility path. That means that if you misuse them today, they probably won't cause a problem, although they may someday later. So, please be very careful only to use the DrawInfo structure between calls to GetScreenDrawInfo() and FreeScreenDrawInfo(), and be sure that you don't forget FreeScreenDrawInfo().

#### BUGS

Does not reflect to changes in screen modes, depth, or pens.

#### SEE ALSO

FreeScreenDrawInfo(), LockPubScreen(), intuition/screens.h

## 1.42 intuition.library/InitRequester

#### NAME

InitRequester -- Initialize a Requester structure.

#### SYNOPSIS

```
InitRequester( Requester )
              A0
```

```
VOID InitRequester( struct Requester * );
```

#### FUNCTION

Initializes a requester for general use. After calling InitRequester, you need fill in only those Requester values that fit your needs.

---

The other values are set to NULL--or zero--states.

Note that the example in the early versions of the Intuition Reference Manual is flawed because the Requester structure is initialized BEFORE InitRequester is called. Be sure to perform your initialization AFTER calling InitRequester.

#### INPUTS

Requester = a pointer to a Requester structure

#### RESULT

None

#### BUGS

Since the publication of the first Intuition Manual to this day, most people haven't used this function, and for compatibility reasons, we'll never be able to assume that they do. Thus, this function is useless.

SEE ALSO

## 1.43 intuition.library/IntuiTextLength

#### NAME

IntuiTextLength -- Return the length (pixel-width) of an IntuiText.

#### SYNOPSIS

```
Length = IntuiTextLength( IText )  
D0                                A0
```

```
LONG IntuiTextLength( struct IntuiText * );
```

#### FUNCTION

This routine accepts a pointer to an instance of an IntuiText structure, and returns the length (the pixel-width) of the string which that instance of the structure represents.

NOTE: if the Font pointer of your IntuiText structure is set to NULL, you'll get the pixel-width of your text in terms of the current system default font. You may wish to be sure that the field IText->ITextFont for 'default font' text is equal to the Font field of the screen it is being measured for.

#### INPUTS

IText = pointer to an instance of an IntuiText structure

#### RESULT

Returns the pixel-width of the text specified by the IntuiText data

#### BUGS

Would do better to take a RastPort as argument, so that a NULL in the Font pointer would lead automatically to the font for the intended target RastPort, rather than the system default font.

SEE ALSO

---

OpenScreen()

## 1.44 intuition.library/ItemAddress

### NAME

ItemAddress -- Returns the address of the specified MenuItem.

### SYNOPSIS

```
Item = ItemAddress( MenuStrip, MenuNumber )
D0                                A0                                D0
```

```
struct MenuItem *ItemAddress( struct Menu *, UWORD );
```

### FUNCTION

This routine feels through the specified menu strip and returns the address of the item specified by the menu number. Typically, you will use this routine to get the address of a menu item from a menu number sent to you by Intuition after user has chosen from a window's menus.

This routine requires that the arguments are well-defined. MenuNumber may be equal to MENUNULL, in which case this routine returns NULL. If MenuNumber doesn't equal MENUNULL, it's presumed to be a valid item number selector for your menu strip, which includes:

- a valid menu number
- a valid item number
- if the item specified by the above two components has a sub-item, the menu number may have a sub-item component, too.

Note that there must be BOTH a menu number and an item number. Because a sub-item specifier is optional, the address returned by this routine may point to either an item or a sub-item.

### INPUTS

MenuStrip = a pointer to the first menu in your menu strip  
MenuNumber = the value which contains the packed data that selects the menu and item (and sub-item). See the Intuition Reference Manual for information on menu numbers.

### RESULT

If MenuNumber == MENUNULL, this routine returns NULL, else this routine returns the address of the menu item specified by MenuNumber.

### BUGS

### SEE ALSO

The "Menus" chapter of the Intuition Reference Manual, or the Amiga Rom Kernel Manual

## 1.45 intuition.library/LockIBase

---

## NAME

LockIBase -- Invoke semaphore arbitration of IntuitionBase.

## SYNOPSIS

```
Lock = LockIBase( LockNumber )
D0                                D0
```

```
ULONG LockIBase( ULONG );
```

## FUNCTION

Grabs Intuition internal semaphore so that caller may examine IntuitionBase safely. This function is not a magic "fix all my race conditions" panacea.

The idea here is that you can get the locks Intuition needs before such IntuitionBase fields as ActiveWindow and FirstScreen are changed, or linked lists of windows and screens are changed.

Do Not Get Tricky with this entry point, and do not hold these locks for long, as all Intuition input processing will wait for you to surrender the lock by a call to UnlockIBase().

NOTE WELL: A call to this function MUST be paired with a subsequent call to UnlockIBase(), and soon, please.

NOTE WELL: Do not call any Intuition functions (nor any graphics, layers, dos, or other high-level system function) while holding this lock.

## INPUTS

A long unsigned integer, LockNumber, specifies which of Intuition's internal locks you want to get. This parameter should be zero for all foreseeable uses of this function, which will let you examine active fields and linked lists of screens and windows with safety.

## RESULT

Returns another ULONG which should be passed to UnlockIBase() to surrender the lock gotten by this call.

## BUGS

This function must not be called while holding any other system locks such as layer or LayerInfo locks.

## SEE ALSO

UnlockIBase(), layers.library/LockLayerInfo(),  
exec.library/ObtainSemaphore()

## 1.46 intuition.library/LockPubScreen

## NAME

LockPubScreen -- Prevent a public screen from closing. (V36)

## SYNOPSIS

```
screen = LockPubScreen( Name )
```

---

D0

A0

```
struct Screen *LockPubScreen( UBYTE * );
```

**FUNCTION**

Prevents a public screen (or the Workbench) from closing while you examine it in preparation of opening a visitor window.

The sequence you use to open a visitor window that needs to examine fields in the screen it is about to open on is:

```
LockPubScreen()
... examine fields ...
OpenWindow() on public screen
UnlockPubScreen()
... use your window ...
CloseWindow()
```

**NOTE**

You needn't hold the "pubscreen lock" for the duration that your window is opened. `LockPubScreen()` basically has the same effect as an open visitor window: it prevents the screen from being closed.

If you pass the string "Workbench" or you pass NULL and there is no default public screen, the Workbench screen will be automatically opened if it is not already present.

**INPUTS**

Name = name string for public screen or NULL for default public screen. The string "Workbench" indicates the Workbench screen.

**RESULT**

Returns pointer to a screen, if successful, else NULL. The call can fail for reasons including that the named public screen doesn't exist or is in private state.

**BUGS****SEE ALSO**

`OpenWindow()`, `UnlockPubScreen()`, `GetScreenData()`

## 1.47 intuition.library/LockPubScreenList

**NAME**

`LockPubScreenList` -- Prevent changes to the system list. (V36)

**SYNOPSIS**

```
List = LockPubScreenList()
D0
```

```
struct List *LockPubScreenList( VOID );
```

**FUNCTION**

Arbitrates access to public screen list while you quickly

make a copy of it for display to the user.

Note that this is intended only for the Public Screen Manager program.

#### NOTES

The nodes on the list are PubScreenNode structures. Act quickly while holding this lock. The restrictions on LockIBase() apply here as well.

#### INPUTS

None.

#### RESULT

A pointer to the public screen list.

#### BUGS

#### SEE ALSO

OpenScreen(), Intuition V36 update documentation

## 1.48 intuition.library/MakeClass

#### NAME

MakeClass -- Create and initialize a boopsi class. (V36)

#### SYNOPSIS

```
iclass = MakeClass( ClassID, SuperClassID, SuperClassPtr,
D0                A0          A1          A2
                  InstanceSize, Flags )
D0                D1
```

```
struct IClass *MakeClass( UBYTE *, UBYTE *, struct IClass *,
                          UWORD, ULONG );
```

#### FUNCTION

For class implementors only.

This function creates a new public or private boopsi class. The superclass should be defined to be another boopsi class: all classes are descendants of the class "rootclass".

Superclasses can be public or private. You provide a name/ID for your class if it is to be a public class (but you must have registered your class name and your attribute ID's with Commodore before you do this!). For a public class, you would also call AddClass() to make it available after you have finished your initialization.

Returns pointer to an IClass data structure for your class. You then initialize the Hook cl\_Dispatcher for your class methods code. You can also set up special data shared by all objects in your class, and point cl\_UserData at it. The last step for public classes is to call AddClass().

You dispose of a class created by this function by calling `FreeClass()`.

#### INPUTS

`ClassID` = NULL for private classes, the name/ID string for public classes  
`SuperClassID` = name/ID of your new class's superclass. NULL if superclass is a private class  
`SuperClassPtr` = pointer to private superclass. Only used if `SuperClassID` is NULL. You are required never to provide a NULL superclass.  
`InstanceSize` = the size of the instance data that your class's objects will require, beyond that data defined for your superclass's objects.  
`Flags` = for future enhancement, including possible additional parameters. Provide zero for now.

#### RESULT

Pointer to the resulting class, or NULL if not possible:  
 - no memory for class data structure  
 - public superclass not found  
 - public class of same name/ID as this one already exists

#### NOTES

#### EXAMPLE

Creating a private subclass of a public class:

```
/* per-object instance data defined by my class      */
struct MyInstanceData {
    ULONG    mid_SomeData;
};

/* some useful table I'll share use for all objects */

UWORD myTable[] = {
    5, 4, 3, 2, 1, 0
};

struct IClass      *
initMyClass()
{
    ULONG __saveds    myDispatcher();
    ULONG    hookEntry();    /* asm-to-C interface glue */
    struct IClass    *cl;
    struct IClass    *MakeClass();

    if ( cl =  MakeClass( NULL,
                        SUPERCLASSID, NULL,    /* superclass is public */
                        sizeof (struct MyInstanceData),
                        0 ))
    {
        /* initialize the cl_Dispatcher Hook */
        cl->cl_Dispatcher.h_Entry = hookEntry;
        cl->cl_Dispatcher.h_SubEntry = myDispatcher;
        cl->cl_Dispatcher.h_Data = (VOID *) 0xFACE; /* unused */
    }
}
```

---



```
        cl-cl_UserData = (ULONG) myTable;
    }
    return ( cl );
}
```

**BUGS**

The typedef 'Class' isn't consistently used. Class pointers used blindly should be APTR, or struct IClass for class implementors.

**SEE ALSO**

FreeClass(), AddClass(), RemoveClass(), NewObject(), Document "Basic Object-Oriented Programming System for Intuition" and the "boopsi Class Reference" document.

## 1.49 intuition.library/MakeScreen

**NAME**

MakeScreen -- Do an Intuition-integrated MakeVPort() of a screen.

**SYNOPSIS**

```
MakeScreen( Screen )
           A0
```

```
VOID MakeScreen( struct Screen * );
```

**FUNCTION**

This procedure allows you to do a MakeVPort() for the viewport of your custom screen in an Intuition-integrated way. This way you can do your own screen manipulations without worrying about interference with Intuition's usage of the same viewport.

The operation of this function is as follows:

- Block until the Intuition View structure is not in being changed.
- Set the view modes correctly to reflect if there is a (visible) interlaced screen.
- call MakeVPort(), passing the Intuition View and your screen's ViewPort.
- Unlocks the Intuition View.

After calling this routine, you should call RethinkDisplay() to incorporate the new viewport of your custom screen into the Intuition display.

NOTE: Intuition may determine that because of a change in global interlace needs that all viewports need to be remade, so it may effectively call RemakeDisplay().

**INPUTS**

Screen = address of the custom screen structure

**RESULT**

None

**BUGS**

SEE ALSO

`RethinkDisplay()`, `RemakeDisplay()`, `graphics.library/MakeVPort()`

## 1.50 intuition.library/ModifyIDCMP

NAME

`ModifyIDCMP` -- Modify the state of a window's IDCMPFlags.

SYNOPSIS

```
[Success =] ModifyIDCMP( Window, IDCMPFlags )
[D0]                A0        D0
```

```
[BOOL] ModifyIDCMP( struct Window *, ULONG );
/* returns BOOL in V37 and greater */
```

FUNCTION

This routine modifies the state of your window's IDCMP (Intuition Direct Communication Message Port). The state is modified to reflect your desires as described by the flag bits in the value IDCMPFlags.

The four actions that might be taken are:

- if there is currently no IDCMP in the given window, and IDCMPFlags is zero, nothing happens
- if there is currently no IDCMP in the given window, and any of the IDCMPFlags is selected (set), then the IDCMP of the window is created, including allocating and initializing the message ports and allocating a signal bit for your port. See the "Input and Output Methods" chapter of the Intuition Reference Manual for full details
- if the IDCMP for the given window exists, and the IDCMPFlags argument is zero, this says that you want Intuition to close the ports, free the buffers and free your signal bit. You MUST be the same task that was active when this signal bit was allocated (either by `ModifyIDCMP()` or `OpenWindow()` ).
- if the IDCMP for the given window is opened, and the IDCMPFlags argument is not zero, this means that you want to change the state of which events will be broadcast to you through the IDCMP

NOTE: You can set up the `Window->UserPort` to any port of your own before you call `ModifyIDCMP()`. If IDCMPFlags is non-null but your UserPort is already initialized, Intuition will assume that it's a valid port with task and signal data preset and Intuition won't disturb your set-up at all, Intuition will just allocate the Intuition message port half of it. The converse is true as well: if UserPort is NULL when you call here with `IDCMPFlags == NULL`, Intuition will deallocate only the Intuition side of the port.

This allows you to use a port that you already have allocated:

- `OpenWindow()` with IDCMPFlags equal to NULL (open no ports)
  - set the UserPort variable of your window to any valid port of your own choosing
  - call `ModifyIDCMP` with IDCMPFlags set to what you want
-

- then, to clean up later, set UserPort equal to NULL before calling CloseWindow() (leave IDCMPFlags alone) BUT FIRST: you must make sure that no messages sent your window are queued at the port, since they will be returned to the memory free pool.

For an example of how to close a window with a shared IDCMP, see the description for CloseWindow().

#### INPUTS

Window = pointer to the Window structure containing the IDCMP ports  
IDCMPFlags = the flag bits describing the new desired state of the IDCMP

#### RESULT

Starting in V37, this function returns NULL if it was unable to create the necessary message ports. (The possibility of failure exists in earlier releases, but no return code was offered). Do not check the return code under V36 or earlier.

#### BUGS

#### SEE ALSO

OpenWindow(), CloseWindow()

## 1.51 intuition.library/ModifyProp

#### NAME

ModifyProp -- Modify the current parameters of a proportional gadget.

#### SYNOPSIS

```
ModifyProp( Gadget, Window, Requester,
            A0      A1      A2
            Flags, HorizPot, VertPot, HorizBody, VertBody )
            D0      D1      D2      D3      D4
```

```
VOID ModifyProp( struct Gadget *, struct Window *,
                 struct Requester *, UWORD, UWORD, UWORD, UWORD );
```

#### FUNCTION

Modifies the parameters of the specified proportional gadget. The gadget's internal state is then recalculated and the imagery is redisplayed in the window or requester that contains the gadget.

The requester variable can point to a requester structure. If the gadget has the GTYP\_REQGADGET flag set, the gadget is in a requester and the window pointer must point to the window of the requester. If this is not the gadget of a requester, the requester argument may be NULL.

NOTE: this function causes all gadgets from the proportional gadget to the end of the gadget list to be refreshed, for reasons of compatibility.

For more refined display updating, use NewModifyProp().

New for V36: ModifyProp() refreshing consists of redrawing gadgets

completely. `NewModifyProp()` has changed this behavior (see `NewModifyProp()`).

#### INPUTS

`PropGadget` = pointer to a proportional gadget  
`Window` = pointer to the window containing the gadget or the window containing the requester containing the gadget.  
`Requester` = pointer to a requester (may be NULL if this isn't a requester gadget)  
`Flags` = value to be stored in the `Flags` field of the `PropInfo`  
`HorizPot` = value to be stored in the `HorizPot` field of the `PropInfo`  
`VertPot` = value to be stored in the `VertPot` field of the `PropInfo`  
`HorizBody` = value to be stored in the `HorizBody` field of the `PropInfo`  
`VertBody` = value to be stored in the `VertBody` field of the `PropInfo`

#### RESULT

None

#### BUGS

#### SEE ALSO

`NewModifyProp()`  
 The Intuition Reference Manual and Amiga Rom Kernel Manual contain more information on Proportional Gadgets.

## 1.52 intuition.library/MoveScreen

#### NAME

`MoveScreen` -- Attempt to move the screen by the increments provided.

#### SYNOPSIS

```
MoveScreen( Screen, DeltaX, DeltaY )
           A0      D0      D1
```

```
VOID MoveScreen( struct Screen *, WORD, WORD );
```

#### FUNCTION

Moves the screen the specified increment, specified in screen pixel resolution coordinates.

New for V36: Screen movement limits have been greatly relaxed, to support screen scrolling. In particular, negative values for screen `LeftEdge` and `TopEdge` may now be valid.

If the `DeltaX` and `DeltaY` variables you specify would move the screen in a way that violates any restrictions, the screen will be moved as far as possible. You may examine the `LeftEdge` and `TopEdge` fields of the `Screen` structure after this function returns to see where the screen really ended up.

In operation, this function determines what the resulting position values that are actually to be used, sets these up, and calls `MakeScreen()` and `RethinkDisplay()`.

#### INPUTS

---

Screen = pointer to a Screen structure  
DeltaX = amount to move the screen on the x-axis  
    Note that DeltaX no longer (V36) need be set to zero  
DeltaY = amount to move the screen on the y-axis  
    Note that these coordinates are in the same resolution  
    as the screen (such as HIRES or INTERLACE)

RESULT  
    None

BUGS

SEE ALSO  
    RethinkDisplay()

## 1.53 intuition.library/MoveWindow

NAME  
    MoveWindow -- Ask Intuition to move a window.

SYNOPSIS  
    MoveWindow( Window, DeltaX, DeltaY )  
                A0          D0          D1  
  
    VOID MoveWindow( struct Window \*, WORD, WORD );

FUNCTION  
    This routine sends a request to Intuition asking to move the window the specified distance. The delta arguments describe how far to move the window along the respective axes.

Note that the window will not be moved immediately, but rather will be moved the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second, and a maximum of sixty times a second.

Interactions with other arbitration of Intuition data structures may defer this operation longer. For V36, you can use the new IDCMP class IDCMP\_CHANGEWINDOW to detect when this operation has completed.

New for V36: Intuition now will do validity checking on the final position. To send absolute movements, or to move and size a window in one step, use ChangeWindowBox().

INPUTS  
    Window = pointer to the structure of the Window to be moved  
    DeltaX = how far to move the Window on the x-axis  
    DeltaY = how far to move the Window on the y-axis

RESULT  
    None

BUGS

---

SEE ALSO

`ChangeWindowBox()`, `SizeWindow()`, `WindowToFront()`, `WindowToBack()`

## 1.54 intuition.library/MoveWindowInFrontOf

NAME

`MoveWindowInFrontOf` -- Arrange the relative depth of a window. (V36)

SYNOPSIS

```
MoveWindowInFrontOf( Window, BehindWindow )
                    A0      A1
```

```
VOID MoveWindowInFrontOf( struct Window *, struct Window * );
```

FUNCTION

Depth-arranges a window in front of an another window.  
Brings out the layers.library `MoveLayerInFrontOf()` to the  
Intuition user.

INPUTS

`Window` = window to re-position in front of another window  
`BehindWindow` = window to re-position in front of

RESULT

Repositions window.

BUGS

Doesn't respect backdrop windows.

SEE ALSO

`WindowToFront()`, `WindowToBack()`, `layers.library/MoveLayerInFrontOf()`

## 1.55 intuition.library/NewModifyProp

NAME

`NewModifyProp` -- `ModifyProp()`, but with selective refresh.

SYNOPSIS

```
NewModifyProp( Gadget, Window, Requester, Flags,
                A0      A1      A2      D0
                HorizPot, VertPot, HorizBody, VertBody, NumGad )
                D1      D2      D3      D4      D5
```

```
VOID NewModifyProp( struct Gadget *, struct Window *,
                    struct Requester *, UWORD, UWORD, UWORD, UWORD, UWORD, WORD );
```

FUNCTION

Performs the function of `ModifyProp()`, but refreshes  
gadgets in the list as specified by the `NumGad` parameter.  
With `NumGad = -1`, this function is identical to `ModifyProp()`.

New for V36: When `NumGad = 1`, this function will now perform

an incremental update of the proportional gadget knob image, rather than refreshing the entire gadget. This means much less flashing when programmatically scrolling a proportional gadget.

#### INPUTS

PropGadget = pointer to a proportional gadget  
 Window = pointer to the window containing the gadget or the window containing the requester containing the gadget.  
 Requester = pointer to a requester (may be NULL if this isn't a requester gadget)  
 Flags = value to be stored in the Flags field of the PropInfo  
 HorizPot = value to be stored in the HorizPot field of the PropInfo  
 VertPot = value to be stored in the VertPot field of the PropInfo  
 HorizBody = value to be stored in the HorizBody field of the PropInfo  
 VertBody = value to be stored in the VertBody field of the PropInfo  
 NumGad = number of gadgets to be refreshed after propgadget internals have been adjusted. -1 means "to end of list."

#### RESULT

None

#### BUGS

#### SEE ALSO

ModifyProp()  
 The Intuition Reference Manual contains more information on Proportional Gadgets.

## 1.56 intuition.library/NewObject

#### NAME

NewObjectA -- Create an object from a class. (V36)  
 NewObject -- Varargs stub for NewObjectA(). (V36)

#### SYNOPSIS

```
object = NewObjectA( class, classID, tagList )
D0                      A0      A1      A2

APTR NewObjectA( struct IClass *, UBYTE *, struct TagItem * );

object = NewObject( class, classID, Tag1, ... )

APTR NewObject( struct IClass *, UBYTE *, ULONG, ... );
```

#### FUNCTION

This is the general method of creating objects from 'boopsi' classes. ('Boopsi' stands for "basic object-oriented programming system for Intuition".)

You specify a class either as a pointer (for a private class) or by its ID string (for public classes). If the class pointer is NULL, then the classID is used.

You further specify initial "create-time" attributes for the

object via a TagItem list, and they are applied to the resulting generic data object that is returned. The attributes, their meanings, attributes applied only at create-time, and required attributes are all defined and documented on a class-by-class basis.

#### INPUTS

class = abstract pointer to a boopsi class gotten via MakeClass().  
 classID = the name/ID string of a public class. This parameter is only used if 'class' is NULL.  
 tagList = pointer to array of TagItems containing attribute/value pairs to be applied to the object being created

#### RESULT

A boopsi object, which may be used in different contexts such as a gadget or image, and may be manipulated by generic functions. You eventually free the object using DisposeObject().

#### NOTES

This function invokes the OM\_NEW "method" for the class specified.

#### BUGS

Typedef's for 'Object' and 'Class' are defined in the include files but not used consistently. The generic type APTR is probably best used for object and class "handles", with the type (UBYTE \*) used for classID strings.

#### SEE ALSO

DisposeObject(), SetAttrs(), GetAttr(), MakeClass(),  
 Document "Basic Object-Oriented Programming System for Intuition"  
 and the "boopsi Class Reference" document.

## 1.57 intuition.library/NextObject

#### NAME

NextObject -- iterate through the object on an Exec list. (V36)

#### SYNOPSIS

```
object = NextObject( objectPtrPtr )
D0                      A0
```

```
APTR NextObject( APTR );
```

#### FUNCTION

This function is for boopsi class implementors only.

When you collect a set of boopsi objects on an Exec List structure by invoking their OM\_ADDMEMBER method, you can (only) retrieve them by iterations of this function.

Works even if you remove and dispose the returned list members in turn.

#### INPUTS

Initially, you set a pointer variable to equal the lh\_Head field of the list (or mlh\_Head field of a MinList).



You pass the *\*address\** of that pointer repeatedly to `NextObject()` until it returns `NULL`.

#### EXAMPLE

```
/* here is the OM_DISPOSE case of some class's dispatcher */
case OM_DISPOSE:
    /* dispose members */
    object_state = mydata->md_CollectionList.lh_Head;
    while ( member_object = NextObject( &object_state ) )
    {
        DoMethod( member_object, OM_REMOVE ); /* remove from list */
        DM( member, msg ); /* and pass along dispose */
    }
```

#### RESULT

Returns pointers to each object in the list in turn, and `NULL` when there are no more.

#### NOTES

#### BUGS

#### SEE ALSO

`DisposeObject()`, `SetAttrs()`, `GetAttr()`, `MakeClass()`,  
Document "Basic Object-Oriented Programming System for Intuition"  
and the "boopsi Class Reference" document.

## 1.58 intuition.library/NextPubScreen

#### NAME

`NextPubScreen` -- Identify next public screen in the cycle. (V36)

#### SYNOPSIS

```
Buff = NextPubScreen( Screen, NameBuff )
D0          A0          A1

UBYTE *NextPubScreen( struct Screen *, UBYTE * );
```

#### FUNCTION

Returns name of next public screen in system rotation, to allow visitor windows to provide function to "jump" among public-screens in a cycle.

#### INPUTS

`Screen` = pointer to the screen your window is currently open in, or `NULL`, if you don't have a pointer to a public screen.  
`NameBuff` = pointer to a buffer of `MAXPUBSCREENNAME` characters, for Intuition to fill in with the name of the next public screen in rotation.

#### RESULT

Returns `NULL` if there are no public screens, otherwise a pointer to your `NameBuff`.

#### NOTES

---

There is no guarantee that the public screen whose name was returned by this function will exist or be in "public" state by the time you call `LockPubScreen()`, etc. You must handle cases where `LockPubScreen()`, etc. will fail.

**BUGS**

The starting screen and cycle order of the public screens isn't defined, so do not draw conclusions about the order you see in the current version of Intuition. We reserve the right to add meaning to the ordering at a future time.

**SEE ALSO**

`OpenScreen()`, Intuition V36 update documentation

## 1.59 intuition.library/ObtainGIRPort

**NAME**

`ObtainGIRPort` -- Set up a RastPort for a custom gadget. (V36)

**SYNOPSIS**

```
RPort = ObtainGIRPort( GInfo )
D0                      A0
```

```
struct RastPort *ObtainGIRPort( struct GadgetInfo * );
```

**FUNCTION**

Sets up a RastPort for use (only) by custom gadget hook routines. This function must be called EACH time a hook routine needing to perform gadget rendering is called, and must be accompanied by a corresponding call to `ReleaseGIRPort()`.

Note that if a hook function passes you a RastPort pointer, e.g., `GM_RENDER`, you needn't call `ObtainGIRPort()` in that case.

**INPUTS**

A pointer to a `GadgetInfo` structure, as passed to each custom gadget hook function.

**RESULT**

A pointer to a RastPort that may be used for gadget rendering. This pointer may be `NULL`, in which case you should do no rendering. You may (optionally) pass a null return value to `ReleaseGIRPort()`.

**BUGS****SEE ALSO**

`ReleaseGIRPort()`, Custom Gadget documentation

## 1.60 intuition.library/OffGadget

**NAME**

`OffGadget` -- Disable the specified gadget.

---

## SYNOPSIS

```
OffGadget( Gadget, Window, Requester )
           A0      A1      A2
```

```
VOID OffGadget( struct Gadget *, struct Window *,
                struct Requester * );
```

## FUNCTION

This command disables the specified gadget. When a gadget is disabled, these things happen:

- its imagery is displayed ghosted
- the GFLG\_DISABLED flag is set
- the gadget cannot be selected by User

The window parameter must point to the window which contains the gadget, or which contains the requester that contains the gadget. The requester parameter must only be valid if the gadget has the GTYP\_REQGADGET flag set, a requirement for all requester gadgets.

NOTE: it's never safe to tinker with the gadget list yourself. Don't supply some gadget that Intuition hasn't already processed in the usual way.

NOTE: for compatibility reasons, this function will refresh all gadgets in a requester, and all gadgets from gadget to the end of the gadget list if gadget is in a window.

If you want to improve on this behavior, you may perform the equivalent operation yourself: remove a gadget or gadgets, change the state of their GFLG\_DISABLED flag, replace the gadgets using AddGLList(), and selectively call RefreshGLList().

## INPUTS

Gadget = pointer to the gadget that you want disabled  
 Window = pointer to a window structure containing the gadget or containing the requester which contains the gadget  
 Requester = pointer to a requester (may be NULL if this isn't a requester gadget (i.e. GTYP\_REQGADGET is not set)).

## RESULT

None

## BUGS

## SEE ALSO

AddGadget(), RefreshGadgets()

## 1.61 intuition.library/OffMenu

## NAME

OffMenu -- Disable the given menu or menu item.

## SYNOPSIS

```
OffMenu( Window, MenuNumber )
```

A0            D0

```
VOID OffMenu( struct Window *, UWORD );
```

#### FUNCTION

This command disables a sub-item, an item, or a whole menu. This depends on the contents of the data packed into MenuNumber, which is described in the Intuition Reference Manual.

#### INPUTS

Window = pointer to the window  
MenuNumber = the menu piece to be disabled

#### RESULT

None

#### BUGS

#### SEE ALSO

OnMenu(), ResetMenuStrip()

## 1.62 intuition.library/OnGadget

#### NAME

OnGadget -- Enable the specified gadget.

#### SYNOPSIS

```
OnGadget( Gadget, Window, Requester )
          A0      A1      A2
```

```
VOID OnGadget( struct Gadget *, struct Window *,
               struct Requester * );
```

#### FUNCTION

This command enables the specified gadget. When a gadget is enabled, these things happen:

- its imagery is displayed normally (not ghosted)
- the GFLG\_DISABLED flag is cleared
- the gadget can thereafter be selected by the user

The window parameter must point to the window which contains the gadget, or which contains the requester that contains the gadget. The requester parameter must only be valid if the gadget has the GTYPE\_REQGADGET flag set, a requirement for all requester gadgets.

NOTE: it's never safe to tinker with the gadget list yourself. Don't supply some gadget that Intuition hasn't already processed in the usual way.

NOTE: for compatibility reasons, this function will refresh all gadgets in a requester, and all gadgets from gadget to the end of the gadget list if gadget is in a window.

If you want to improve on this behavior, you may perform the equivalent operation yourself: remove a gadget or gadgets,

change the state of their GFLG\_DISABLED flag, replace the gadgets using AddGList(), and selectively call RefreshGList().

#### INPUTS

Gadget = pointer to the gadget that you want disabled  
Window = pointer to a window structure containing the gadget or containing the requester which contains the gadget  
Requester = pointer to a requester (may be NULL if this isn't a requester gadget (i.e. GTYP\_REQGADGET is not set)).

#### RESULT

None

#### BUGS

#### SEE ALSO

## 1.63 intuition.library/OnMenu

#### NAME

OnMenu -- Enable the given menu or menu item.

#### SYNOPSIS

```
OnMenu( Window, MenuNumber )
        A0      D0
```

```
VOID OnMenu( struct Window *, UWORD );
```

#### FUNCTION

This command enables a sub-item, an item, or a whole menu. This depends on the contents of the data packed into MenuNumber, which is described in the Intuition Reference Manual.

#### INPUTS

Window = pointer to the window  
MenuNumber = the menu piece to be enables

#### RESULT

None

#### BUGS

#### SEE ALSO

OffMenu(), ResetMenuStrip()

## 1.64 intuition.library/OpenScreen

#### NAME

OpenScreen -- Open an Intuition screen.

#### SYNOPSIS

```
Screen = OpenScreen( NewScreen )
```

---

```

D0                                A0

struct Screen *OpenScreen( struct NewScreen * );

or

struct Screen *OpenScreen( struct ExtNewScreen * );

```

#### FUNCTION

Opens an Intuition screen according to the specified parameters found in the NewScreen structure.

Does all the allocations, sets up the screen structure and all substructures completely, and links this screen's viewport into Intuition's View structure.

Before you call OpenScreen(), you must initialize an instance of a NewScreen structure. NewScreen is a structure that contains all of the arguments needed to open a screen. The NewScreen structure may be discarded immediately after OpenScreen() returns.

The SHOWTITLE flag is set to TRUE by default when a screen is opened. To change this, you must call the routine ShowTitle().

#### INPUTS

NewScreen = pointer to an instance of a NewScreen structure.

New for V36:

In addition to the information contained in the NewScreen structure, Intuition now recognizes extended data passed in the form of an array of TagItem structures (from <utility/tagitem.h>), commonly called a "tag list."

There are two ways to provide this array. The first is to use the new Intuition entry point OpenScreenTagList() and pass the tag list as a parameter. This is the recommended method, and has a convenient format variation for C programs using a variable number of arguments.

An older way used for some V36 development uses the OpenScreen() entry point, and an extension of the NewScreen structure named ExtNewScreen. See the documentation of the flag NS\_EXTENDED, below.

While we recommend that you use OpenScreenTagList() rather than OpenScreen() when using the extension tag list, we document the tag ID values here, so that all parameters for opening a screen can be found in one place.

NewScreen is initialized with the following information:

-----  
Left = initial x-position of your screen (should be zero for releases prior to V36)

Top = initial y-position of the opening screen  
(Note: Left and Top are specified relative to the Intuition's view, in same resolution as the screen pixels.)

---

Width = the width for this screen's RastPort

Height = the height for this screen's RastPort, or the constant STDSCREENHEIGHT to get the current default height (at this time guaranteed to be at least 200 rows). The normal width and height for a particular system is stored by the graphics.library in GfxBase->NormalDisplayRows and GfxBase->NormalDisplayColumns. These values will be different depending on factors such as PAL video and overscan.

For V36, a new constant STDSCREENWIDTH is introduced. It serves the similar function for screen width. Both STDSCREENWIDTH and STDSCREENHEIGHT indicate that your screen RastPort is to be the same dimensions as your DisplayClip rectangle. If you do not specify either a standard or custom DisplayClip, the OSCAN\_TEXT region will be used, which corresponds to the standard dimensions of V35 and earlier.

Furthermore, if you are using OpenScreenTagList(), and you specify STDSCREENWIDTH, and you DO NOT provide a NewScreen pointer, and you DO NOT provide SA\_Left, then Intuition will automatically set the LeftEdge of the screen to be the left edge of the screen's DisplayClip region. Likewise for STDSCREENHEIGHT and the screen's TopEdge.

Depth = number of bitplanes

DetailPen = pen number for details (like gadgets or text in title bar)  
The common value for this pen is 0.

BlockPen = pen number for block fills (like title bar)  
The common value for this pen is 1.

Type = screen type values

Set these flags as desired from the set:

CUSTOMSCREEN -- this is your own screen, not a system screen.

CUSTOMBITMAP -- this custom screen has bit maps supplied in the bitmap field of the NewScreen structure. Intuition is not to allocate any raster bitmaps.

SCREENBEHIND -- your screen will be created behind all other open screens. This allows a program to prepare imagery in the screen, change its colors, and so on, bringing it to the front when it is presentable.

SCREENQUIET -- Intuition will not render system screen gadgets or screen title. In concert with the WFLG\_RMBTRAP flag on all your screen's windows, this flag will prevent Intuition from rendering into your screen's bitplanes. Without WFLG\_RMBTRAP (or using the IDCMP\_MENUVERIFY facility to cancel menu operations), this flag will prevent Intuition from clearing your menu bar, which is probably unacceptable. The menu bar layer may still overwrite a portion of your screen bitmap when the screen is opened. (V36: it won't clobber your bits any more.)

NS\_EXTENDED for this screen to use extended attributes pointed to by the 'Extended' field, below.

---

ViewModes = the appropriate argument for the data type ViewPort.Modes.  
These include:

HIRES for this screen to be HIRES width.

INTERLACE for the display to switch to interlace.

SPRITES for this screen to use sprites (the pointer  
sprite is always displayed)

DUALPF for dual-playfield mode (not supported yet)

[For V36: The ViewModes field is superceded by a TagItem with  
tag value SA\_DisplayID.]

Font = pointer to the default TextAttr structure for text in this  
screen and all windows that open in this screen. Text that uses  
this TextAttr includes title bars of both screen and windows,  
string gadgets, and menu titles. Of course, IntuiText that  
specifies a NULL TextAttr field will use the screen/window default  
fonts. NOTE: Intuition will \*NOT\* call OpenDiskFont(), so  
the TextAttr you supply must be in memory. The ways to ensure  
that are to either use a ROM font (Topaz 8 or 9) or first  
call OpenDiskFont() to load the font, and don't close it  
until after your screen is successfully opened.

[For V36: this is superceded by SA\_Font and SA\_SysFont.]

DefaultTitle = pointer to a line of text that will be displayed along  
the screen's title bar. Null terminated, or just a NULL pointer  
to get no text

[For V36: superceded by SA\_Title.]

Gadgets = This field should be set to NULL, since no user gadgets may  
be attached to a screen with the current versions of Intuition.

CustomBitMap = if you're not supplying a custom bitmap, this value is  
ignored. However, if you have your own display memory that you  
want used for this screen, the CustomBitMap field should point to  
the BitMap structure that describes your display memory. See the  
"Screens" chapter and the "Amiga ROM Kernel Manual" for more  
information about bitmaps.

[For V36: this is superceded by SA\_BitMap.]

[ All TagItem extensions below are new for V36.]

Extension = if NS\_EXTENDED is set in NewScreen.Type, this pointer  
should point to an array (or chain of arrays) of TagItems,  
as defined in the include file <utility/tagitem.h>. This  
field is only defined in the structure ExtNewScreen.  
The values to use for TagItem.ti\_Tag are defined below. We  
recommend that V36-specific applications use the new Intuition  
entry point OpenScreenTagList(), rather than using this field.  
The ExtNewScreen structure is a convenient way to give V36  
Intuition some information that V34 and earlier Intuition will  
ignore.

Each TagItem is an optional tagged data structure which identifies  
an additional parameter to OpenScreen(). The applicable tag ID  
values for TagItem.ti\_Tag and their corresponding data follow.

Several of the tag items are alternative (and overriding) versions  
to familiar fields in NewScreen. They are:



SA\_Left  
SA\_Top  
SA\_Width  
SA\_Height  
SA\_Depth  
SA\_DetailPen  
SA\_BlockPen  
SA\_Title  
SA\_Font  
SA\_Type  
SA\_BitMap (whose existence also implies CUSTOMBITMAP).

Several tags are Booleans, which means that depending on whether their corresponding ti\_Data field is zero (FALSE) or non-zero (TRUE), they specify Boolean attributes. The ones corresponding to Boolean flags in the NewScreen.Type field are:

SA\_ShowTitle  
SA\_Behind (equiv. to SCREENBEHIND)  
SA\_Quiet (equiv. to SCREENQUIET)

The following tags provide extended information to Intuition when creating a screen:

SA\_DisplayID: ti\_Data is a 32-bit extended display mode ID as defined in <graphics/displayinfo.h>

SA\_Overscan: ti\_Data contains a defined constant specifying one of the system standard overscan dimensions appropriate for the display mode of the screen. Used with the Width and Height dimensions STDSCREENWIDTH and STDSCREEN, this makes it trivial to open an overscanned or standard dimension screen. You may also hand-pick your various dimensions for overscanned or other screens, by specifying screen position and dimensions explicitly, and by using SA\_DClip to explicitly specify an overscanned DisplayClip region.

The values for ti\_Data of this tag are as follows:

OSCAN\_TEXT - Text Overscan region. A region which is completely on screen and readable ("text safe"). A preferences data setting, this is backward equivalent with the old MoreRows, and specifies the DisplayClip and default dimensions of the Workbench screen. This is the default.

OSCAN\_STANDARD - Also a preferences setting, this specifies a rectangle whose edges are "just out of view." This yields the most efficient position and dimensions of on-monitor presentations, such as games and artwork.

OSCAN\_MAX - This is the largest rectangular region that the graphics library can handle "comfortably" for a given mode. Screens can smoothly scroll (hardware pan) within this region, and any DisplayClip or Screen region within this rectangle is also legal. It is not a preferences item, but reflects the limits of the graphics hardware and software.

---

OSCAN\_VIDEO - This is the largest region that the graphics library can display, comfortable or not. There is no guarantee that all smaller rectangles are valid. This region is typically out of sight on any monitor or TV, but provides our best shot at "edge-to-edge" video generation.

Remember, using overscan drastically effects memory use and chip memory bandwidth. Always use the smallest (standard) overscan region that works for your application.

SA\_DClip: ti\_Data is a pointer to a rectangle which explicitly defines a DisplayClip region for this screen. See QueryOverscan() for the role of the DisplayClip region.

Except for overscan display screens, this parameter is unnecessary, and specifying a standard value using SA\_Overscan is normally an easier way to get overscan.

SA\_AutoScroll: this is a Boolean tag item, which specifies that this screens is to scroll automatically when the mouse pointer reaches the edge of the screen. The operation of this requires that the screen dimensions be larger than its DisplayClip region.

SA\_PubName: If this field is present (and ti\_Data is non-NULL), it means that the screen is a public screen, and that the public screen name string is pointed to by ti\_Data. Public screens are opened in "PRIVATE" mode and must be made public using PubScreenStatus().

SA\_Pens: The ti\_Data field (if non-NULL) points to a UWORD array of pen specification, as defined for struct DrawInfo. This array will be used to initialize the screen's DrawInfo.dri\_Pens array.

SA\_Pens is also used to decide that a screen is ready to support the full-blown "new look" graphics. If you want the 3D embossed look, you must provide this tag, and the ti\_Data value cannot be NULL. If it points to a "minimal" array, containing just the terminator ~0, you can specify "new look" without providing any values for the pen array.

The following two tag items specify the task and signal to be issued to notify when the last "visitor" window closes on a public screen. This support is to assist envisioned public screen manager programs.

SA\_PubTask: Task to be signalled. If absent (and SA\_PubSig is valid), use the task which called OpenScreen() or OpenScreenTagList()).

SA\_PubSig: Data is a UBYTE signal number (not flag) used to notify a task when the last visitor window closes on a public screen.

SA\_Colors: ti\_Data points to an array of ColorSpec structures (terminated with ColorIndex = -1) which specify initial

values of the screen's color palette.

**SA\_FullPalette:** this is a Boolean attribute. Prior to V36, there were just 7 RGB color values that Intuition maintained in its user preferences (playfield colors 0-3, and colors 17-19 for the sprite). When opening a screen, the color map for the screens viewport is first initialized by graphics (graphics.library/GetColorMap()) then these seven values are overridden to take the preferences values.

In V36, Intuition maintains a full set of 32 preferences colors. If you specify TRUE for SA\_FullPalette, Intuition will override ALL color map entries with its full suite of preferred colors.

**SA\_ErrorCode:** ti\_Data points to a ULONG in which Intuition will stick an extended error code if OpenScreen[TagList]() fails. Values are of this include 0, for success, and:

OSERR_NOMONITOR	- monitor for display mode not available.
OSERR_NOCHIPS	- you need newer custom chips for display mode.
OSERR_NOMEM	- couldn't get normal memory
OSERR_NOCHIPMEM	- couldn't get chip memory
OSERR_PUBNOTUNIQUE	- public screen name already used
OSERR_UNKNOWNMODE	- don't recognize display mode requested

NOTE: These values are not the same as some similar return values defined in graphics.library/ModeNotAvailable().

**SA\_SysFont:** ti\_Data selects one of the system standard fonts specified in preferences. This tag item overrides the NewScreen.Font field and the SA\_Font tag item.

Values recognized in ti\_Data at present are:

- 0 - old DefaultFont, fixed-width, the default.
- 1 - Workbench screen preferred font. You have to be very font sensitive to handle a proportional or larger than traditional screen font.

NOTE WELL: if you select sysfont 1, windows opened on your screen will not inherit the screen font, but rather the window RastPort will be initialized to the old-style DefaultFont (sysfont 0).

#### RESULT

If all is well, returns the pointer to your new screen  
 If anything goes wrong, returns NULL, with further error specification in the variable pointed to by the SA\_ErrorCode data field (V36 and later).

#### NOTE

By default, AmigaDOS requesters related to your process are put on the Workbench screen (these are messages like "Disk Full"). If you wish them to show up on custom screens, DOS must be told. This fragment shows the procedure. More information is available in the AmigaDOS manuals. Sample code fragment:

```
#include "libraries/dosextens.h"
```

```

...
struct Process    *process;
struct Window     *window;
APTR              temp;

...
process = (struct Process *) FindTask(NULL);
temp = process->pr_WindowPtr;    (save old value)
process->pr_WindowPtr = (APTR) window;
( use a pointer to any open window on your screen )

...
your code goes here

...
process->pr_WindowPtr = temp;
( restore value BEFORE CloseWindow() )
CloseWindow(window);

```

BUGS

SEE ALSO

OpenScreenTagList(), OpenWindow(), PrintIText(), CloseScreen(),  
 QueryOverScan() PubScreenStatus(), The Intuition Reference Manual,  
 utility/tagitem.h, graphics.library/ModeNotAvailable(),  
 diskfont.library/OpenDiskFont(), graphics.library/GetColorMap()

## 1.65 intuition.library/OpenScreenTagList

NAME

OpenScreenTagList -- OpenScreen() with TagItem extension array. (V36)  
 OpenScreenTags -- Varargs stub for OpenScreenTagList. (V36)

SYNOPSIS

```

Screen = OpenScreenTagList( NewScreen, TagItems )
D0                      A0          A1

struct Screen *OpenScreenTagList( struct NewScreen *,
                                  struct TagItem * );

Screen = OpenScreenTags( NewScreen, Tag1, ... )

struct Screen *OpenScreenTags( struct NewScreen *,
                               ULONG, ... );

```

FUNCTION

Provides an extension to the parameters passed to OpenScreen(). This extensions is in the form of (a pointer to) an array of TagItem structures, which have to fields: ti\_Tag, an ID identifying the meaning of the other field, ti\_Data. See <utility/tagitem.h>.

The tag items can supplement or override the values in NewScreen. In fact, you can pass a NULL value of the NewScreen pointer. For that matter, if you pass NULL in both arguments, you'll get a screen with defaults in all fields, including display mode, depth, colors, dimension, title, and so on. We ask that you at least supply a title when you open a screen.

See `OpenScreen()` documentation for parameter specifications.

#### INPUTS

`NewScreen` - (optional) pointer to a `NewScreen` structure.  
`TagItems` - (optional) pointer to (an array of) `TagItem` structures, terminated by the value `TAG_END`.

#### RESULT

`Screen` - an open Intuition screen. See `OpenScreen()` for extended error codes when `Screen` is returned `NULL`.

#### EXAMPLE

The version using a variable number of arguments must be created for each particular compiler, and may not have an analogue in all versions. For vanilla, 32-bit C parameter passing conventions, this works (and will appear in `amiga.lib`):

```
struct Screen      *
OpenScreenTags( ns, tag1 )
struct NewScreen   *ns;
ULONG              tag1;
{
    struct Screen   *OpenScreenTagList();

    return ( OpenScreenTagList( ns, (struct TagItem *) &tag1 ) );
}
```

#### NOTES

We recommend this extension to `OpenScreen()` over using the field `ExtNewScreen.Extension`. However, the `ExtNewScreen.Extension` is a convenient way to supply a few tags to V36 Intuition which will be ignored by V34 Intuition. See `OpenScreen()` documentation for lots of details.

#### BUGS

#### SEE ALSO

`OpenScreen()`

## 1.66 intuition.library/OpenWindow

#### NAME

`OpenWindow` -- Open an Intuition window.

#### SYNOPSIS

```
Window = OpenWindow( NewWindow )
D0                      A0
```

```
struct Window *OpenWindow( struct NewWindow * );
```

#### FUNCTION

Opens an Intuition window of the given dimensions and position, with the properties specified in the `NewWindow` structure. Allocates everything you need to get going.

New for V36: there is an extensive discussion of public Screens and visitor windows at the end of this section. Also, you can provide extensions to the NewWindow parameters using an array of TagItem structures. See the discussion below, and the documentation for the function OpenScreenTagList().

Before you call OpenWindow(), you must initialize an instance of a NewWindow structure. NewWindow is a structure that contains all of the arguments needed to open a window. The NewWindow structure may be discarded immediately after it is used to open the window.

If Type == CUSTOMSCREEN, you must have opened your own screen already via a call to OpenScreen(). Then Intuition uses your screen argument for the pertinent information needed to get your window going. On the other hand, if type == one of the Intuition's standard screens, your screen argument is ignored. Instead, Intuition will check to see whether or not that screen already exists: if it doesn't, it will be opened first before Intuition opens your window in the standard screen.

New for V36: If you specify Type == WBENCHSCREEN, then your window will appear on the Workbench screen, unless the global public screen mode SHANGHAI is set, in which case your window will be "hijacked" to the default public screen. See also SetPubScreenModes().

New for V36: If the WFLG\_NW\_EXTENDED flag is set, it means that the field 'ExtNewWindow->Extension' points to an array of TagItems, as defined in intuition/tagitem.h. This provides an extensible means of providing extra parameters to OpenWindow. For compatibility reasons, we could not add the 'Extension' field to the NewWindow structure, so we have defined a new structure ExtNewWindow, which is identical to NewWindow with the addition of the Extension field.

We recommend that rather than using ExtNewWindow.Extension, you use the new Intuition function OpenWindowTagList() and its varargs equivalent OpenWindowTags(). We document the window attribute tag ID's (ti\_Tag values) here, rather than in OpenWindowTagList(), so that you can find all the parameters for a new window defined in one place.

If the WFLG\_SUPER\_BITMAP flag is set, the bitmap variable must point to your own bitmap.

The DetailPen and the BlockPen are used for system rendering; for instance, the title bar is first filled using the BlockPen, and then the gadgets and text are rendered using DetailPen. You can either choose to supply special pens for your window, or, by setting either of these arguments to -1, the screen's pens will be used instead.

Note for V36: The DetailPen and BlockPen no longer determine what colors will be used for window borders, if your window opens on a "full-blown new look screen."

## INPUTS

---

NewWindow = pointer to an instance of a NewWindow structure. That structure is initialized with the following data:

---

Left = the initial x-position for your window

Top = the initial y-position for your window

Width = the initial width of this window

Height = the initial height of this window

DetailPen = pen number (or -1) for the rendering of window details (like gadgets or text in title bar)

BlockPen = pen number (or -1) for window block fills (like title bar) [For V36: Title bar colors are determined otherwise.]

Flags = specifiers for your requirements of this window, including: which system gadgets you want attached to your window:

- WFLG\_DRAGBAR allows this window to be dragged
- WFLG\_DEPTHGADGET lets the user depth-arrange this window
- WFLG\_CLOSEGADGET attaches the standard close gadget
- WFLG\_SIZEGADGET allows this window to be sized.

If you ask for the WFLG\_SIZEGADGET gadget, you must specify one or both of the flags WFLG\_SIZEEBRIGHT and WFLG\_SIZEEBOTTOM below; if you don't, the default is WFLG\_SIZEEBRIGHT. See the following items WFLG\_SIZEEBRIGHT and WFLG\_SIZEEBOTTOM for more details.

- WFLG\_SIZEEBRIGHT is a special system gadget flag that you set to specify whether or not you want the RIGHT border adjusted to account for the physical size of the sizing gadget. The sizing gadget must, after all, take up room in either the right or bottom border (or both, if you like) of the window. Setting either this or the WFLG\_SIZEEBOTTOM flag selects which edge will take up the slack. This will be particularly useful to applications that want to use the extra space for other gadgets (like a proportional gadget and two Booleans done up to look like scroll bars) or, for instance, applications that want every possible horizontal bit and are willing to lose lines vertically. NOTE: if you select WFLG\_SIZEGADGET, you must select either WFLG\_SIZEEBRIGHT or WFLG\_SIZEEBOTTOM or both. If you select neither, the default is WFLG\_SIZEEBRIGHT.
- WFLG\_SIZEEBOTTOM is a special system gadget flag that you set to specify whether or not you want the BOTTOM border adjusted to account for the physical size of the sizing gadget. For details, refer to WFLG\_SIZEEBRIGHT above.

- WFLG\_GIMMEZEROZERO for easy but expensive output

what type of window layer you want, either:

- WFLG\_SIMPLE\_REFRESH
- WFLG\_SMART\_REFRESH
- WFLG\_SUPER\_BITMAP

- WFLG\_BACKDROP for whether or not you want this window to be one of Intuition's special backdrop windows. See WFLG\_BORDERLESS
-

as well.

- `WFLG_REPORTMOUSE` for whether or not you want to "listen" to mouse movement events whenever your window is the active one. After you've opened your window, if you want to change you can later change the status of this via a call to `ReportMouse()`. Whether or not your window is listening to mouse is affected by gadgets too, since they can cause you to start getting reports too if you like. The mouse move reports (either `InputEvents` or messages on the `IDCMP`) that you get will have the x/y coordinates of the current mouse position, relative to the upper-left corner of your window (`WFLG_GIMMEZEROZERO` notwithstanding). This flag can work in conjunction with the `IDCMP` Flag called `IDCMP_MOUSEMOVE`, which allows you to listen via the `IDCMP`.
- `WFLG_BORDERLESS` should be set if you want a window with no border padding. Your window may have the border variables set anyway, depending on what gadgetry you've requested for the window, but you won't get the standard border lines and spacing that comes with typical windows.

This is a good way to take over the entire screen, since you can have a window cover the entire width of the screen using this flag. This will work particularly well in conjunction with the `WFLG_BACKDROP` flag (see above), since it allows you to open a window that fills the ENTIRE screen. NOTE: this is not a flag that you want to set casually, since it may cause visual confusion on the screen. The window borders are the only dependable visual division between various windows and the background screen. Taking away that border takes away that visual cue, so make sure that your design doesn't need it at all before you proceed.

- `WFLG_ACTIVATE` is the flag you set if you want this window to automatically become the active window. The active window is the one that receives input from the keyboard and mouse. It's usually a good idea to have the window you open when your application first starts up be an `ACTIVATED` one, but all others opened later not be `ACTIVATED` (if the user is off doing something with another screen, for instance, your new window will change where the input is going, which would have the effect of yanking the input rug from under the user). Please use this flag thoughtfully and carefully.

Some notes: First, your window may or may not be active by the time this function returns. Use the `IDCMP_ACTIVEWINDOW` `IDCMP` message to know when your window has become active. Also, be very careful not to mistakenly specify the obsolete flag names `WINDOWACTIVE` or `ACTIVEWINDOW`. These are used in other contexts, and their values unintentionally added to your flags can cause most unfortunate results. To avoid confusion, they are now known as `WFLG_WINDOWACTIVE` and `IDCMP_ACTIVEWINDOW`.

- `WFLG_RMBTRAP`, when set, causes the right mouse button events
-



to be trapped and broadcast as events. You can receive these events through either the IDCMP or the console.

- `WFLG_NOCAREREFRESH` indicates that you do not wish to be responsible for calling `BeginRefresh()` and `EndRefresh()` when your window has exposed regions (i.e., when the `IDCMP_REFRESHWINDOW` message would be generated). See also the descriptions of these two functions.
- `WFLG_NW_EXTENDED` (V36) indicates that `NewWindow` in fact points to an `ExtNewWindow` structure, and that the 'Extension' field points to an array of `TagItem` structures, with meaning described below.

`IDCMPFlags = IDCMP` is the acronym for Intuition Direct Communications Message Port. (It's Intuition's sole acronym.) If any of the IDCMP Flags is selected, Intuition will create a pair of message ports and use them for direct communications with the task opening this window (as compared with broadcasting information via the Console device). See the "Input and Output Methods" chapter of the Intuition Reference Manual for complete details.

You request an IDCMP by setting any of these flags. Except for the special `VERIFY` flags, every other flag you set tells Intuition that if a given event occurs which your program wants to know about, it is to broadcast the details of that event through the IDCMP rather than via the Console device. This allows a program to interface with Intuition directly, rather than going through the Console device.

Many programs have elected to use IDCMP communication exclusively, and not to associate a console with their windows at all. Some operations, such as `IDCMP_MENUVERIFY`, can ONLY be achieved using IDCMP.

The IDCMP flags you can set are:

- `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.

With all the "VERIFY" functions, it is not save to leave them enabled at any time when your task may not be able to respond for a long period.

It is NEVER safe to call AmigaDOS, directly or indirectly, when a "VERIFY" function is active. If AmigaDOS needs to put up a disk requester for you, your task might end up waiting for the requester to be satisfied, at the same time as Intuition is waiting for your response. The result is a complete machine lockup. USE ModifyIDCMP() TO TURN OFF ANY VERIFY MESSAGES BEFORE CALLING dos.library!!

For V36: If you do not respond to the verification IntuiMessages within the user specified timeout duration, Intuition will abort the operation. This eliminates the threat of these easy deadlocks, but can result in a confused user. Please try hard to continue to avoid "logical deadlocks".

- 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.

NOTE 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.

---

- 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...

Gadgets = the pointer to the first of a linked list of the your own Gadgets which you want attached to this Window. Can be NULL if you have no Gadgets of your own

CheckMark = a pointer to an instance of the struct Image where can be found the imagery you want used when any of your menu items is to be checkmarked. If you don't want to supply your own imagery and you want to just use Intuition's own checkmark, set this argument to NULL

Text = a null-terminated line of text to appear on the title bar of your window (may be null if you want no text)

Type = the screen type for this window. If this equal CUSTOMSCREEN, you must have already opened a CUSTOMSCREEN (see text above). Types available include:

- WBENCHSCREEN
- CUSTOMSCREEN
- PUBLICSCREEN (new for V36, see text below)

Screen = if your type is one of Intuition's standard screens, then this argument is ignored. However, if Type == CUSTOMSCREEN, this must point to the structure of your own screen

BitMap = if you have specified WFLG\_SUPER\_BITMAP as the type of refreshing you want for this window, then this value points to a instance of the struct bitmap. However, if the refresh type is NOT WFLG\_SUPER\_BITMAP, this pointer is ignored.

MinWidth, MinHeight, MaxWidth, MaxHeight = the size limits for this window. These must be reasonable values, which is to say that the minimums cannot be greater than the current size, nor can the maximums be smaller than the current size. If they are, they're ignored. Any one of these can be initialized to zero, which means that that limit will be set to the current dimension of that axis. The limits can be changed after the Window is opened by calling the WindowLimits() routine.

NOTE: ORIGINALLY, we stated that:

"If you haven't requested the WFLG\_SIZEGADGET option, these variables are ignored so you don't have to initialize them."

It is now clear that a variety of programs take it upon themselves to call SizeWindow() (or ChangeWindowBox()) without your program's consent or consulting your WFLG\_SIZEGADGE option. To protect yourself against the results, we strongly urge that if you supply suitable values for these fields even if you do not specify WFLG\_SIZEGADGET.

The maximums may be LARGER than the current size, or even larger than the current screen. The maximums should be set to

the highest value your application can handle. This allows users with larger display devices to take full advantage of your software. If there is no good reason to limit the size, then don't. -1 or ~0 indicates that the maximum size is only limited by the size of the window's screen.

See also the docs on the function `WindowLimits()` for more information.

Extension (New for V36) = a pointer to an array (or chain of arrays) of `TagItems` to specify additional parameters to `OpenWindow()`. `TagItems` in general are described in `utility/tagitem.h`, and the `OpenWindow` tags are defined in `intuition/intuition.h` and described here. For items pertaining to Public Screens and visitor windows, please see below.

Here are the `TagItem.ti_Tag` values that are defined for `OpenWindow` (and `OpenWindowTagList()`).

Certain tags simply override equivalent values in `NewWindow`, and allow you to open a window using `OpenWindowTagList()` without having a `NewWindow` structure at all. In each case, cast the corresponding data to `ULONG` and put it in `ti_Data`.

The compatible tag items include:

```
WA_Left
WA_Top
WA_Width
WA_Height
WA_DetailPen
WA_BlockPen
WA_IDCMP
WA_Flags           - initial values for Flags before looking at other
                   Boolean component Tag values
WA_Gadgets
WA_Checkmark
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_SIZEEBRIGHT
WA_SizeBBottom	- equivalent to WFLG_SIZEEBOTTOM
WA_GimmeZeroZero	- equivalent to WFLG_GIMMEZEROZERO

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 (default FALSE) 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.

**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 is 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.

`WA_MouseQueue`

`WA_RptQueue` - These two tags specify limits for the number of outstanding `IntuiMessages` that `Intuition` will send for `IDCMP_MOUSEMOVE` and repeated `IDCMP_RAWKEY`, respectively. You can change the value of the mouse queue limit after the window is open using `SetMouseQueue()`, but there is not currently an equivalent function for 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()`.

## NOTES

Regarding Public Screens, you can specify a window to be a "visitor window" on a public screen in one of several ways. In each case, you must be sure not to specify a `NewWindow` type of `CUSTOMSCREEN`. You should use the value `PUBLICSCREEN`.

There are actually several ways you can specify which screen you want a visitor window to be opened on:

- 1) Specify the name of the public screen `WA_PubScreenName`, or a `NULL` pointer, in `ti_Data`. The name might have been provided by the user. A `NULL` pointer means to use the default public screen.

If the named screen cannot be found, the default public screen will be used if the Boolean attribute `WA_PubScreenFallBack` is `TRUE`.

- 2) Specify a pointer to a public screen using the `WA_PubScreen` tag item. The `WA_PubScreenFallBack` attribute has no effect. You can specify the default



public screen by providing a NULL pointer.

You can also specify the pointer by setting `NewWindow.Type` to `PUBLICSCREEN`, and specifying the public screen pointer in `NewWindow.Screen`. The `WA_PubScreen` tag item has precedent over this technique.

Unless `NULL`, the screen pointer provided **MUST** be a valid public screen. You may ensure this several ways:

- Be the owner of the screen.
- Have a window already open on the screen.
- Use `LockPubScreen()` to prevent the screen from closing.
- specifying the `WFLG_VISITOR` bit in `NewWindow.Flags` is not supported.

It is anticipated that the last will be the most common method of opening public screens because you often want to examine properties of the screen your window will be using in order to compensate for differences in dimension, depth, and font.

The standard sequence for this method is as follows:

```
LockPubScreen() - obtain a pointer and a promise
layout window   - adapt your window to the screen you will use
OpenWindow()    - using the pointer you specify
UnlockPubScreen() - once your window is open, you can let go
                  of the lock on the public screen
... normal window even processing ...
CloseWindow().
```

Regarding "service" windows, such as those opened for a system requester or file requester associated with a given "client" window. These windows should NOT be "visitor" windows. Open them using `NewWindow.Type = CUSTOMSCREEN` and `NewWindow.Screen` equal to the screen of the client window (`window->WScreen`). You can also use `WA_CustomScreen`, which has precedence.

This ensures that the requester service window will be allowed to open on the same screen as the client window, even if that screen is not a public screen, or has private status.

This has an implication for service/client protocol: when you pass a window pointer to any system requester routine or to a routine which creates some other other service window, you **MUST** keep your window open until the client window is closed.

If a requester service will allow a `NULL` client window, this should indicate to open the service window on the default public screen (probably `Workbench`). The correct way to get a pointer to this screen is to call `LockPubScreen( NULL )`. In this case, you want to open as a visitor window, which means you should use either `PUBLICSCREEN` or `WA_PubScreen`, described above. You should call `UnlockPubScreen()` after your visitor window is open.

As of V36, gadgets in the right and bottom border (specified with `GACT_RIGHTBORDER` and `GACT_BOTTOMBORDER`) only

---

contribute to the dimensions of the borders if they are also GFLG\_RELRIGHT and GFLG\_RELBOTTOM, respectively.

#### RESULT

If all is well, returns the pointer to your new Window  
If anything goes wrong, returns NULL

#### BUGS

When you open a window, Intuition will set the font of the window's RastPort to the font of the window's screen. This does not work right for GimmeZeroZero windows: the BorderRPort RastPort has the font set correctly, but Window.RPort is set up with the system default font. For compatibility reasons, we won't be fixing this problem.

Also, there is a compatibility trick going on with the default font of your window's RastPort if the screen's font is "fancy." See the SA\_SysFont attribute described under OpenScreen().

Unless you arrange otherwise, each window you open will allocate a signal for your task from the 16 "user signals." If no signal is available, your window will not be able to be opened. In early V36 versions and before, Intuition didn't check this condition, but just left you with an unusable port.

#### SEE ALSO

OpenWindowTagList(), OpenScreen(), ModifyIDCMP(), SetWindowTitles(), LockPubScreen(), SetDefaultPubScreen(), ZipWindow(), layers.library/InstallLayerHook(), SetPubScreenModes()

## 1.67 intuition.library/OpenWindowTagList

#### NAME

OpenWindowTagList -- OpenWindow() with TagItem extension. (V36)  
OpenWindowTags -- Varargs stub for OpenWindowTagList (V36)

#### SYNOPSIS

```
Window = OpenWindowTagList( NewWindow, TagItems )
D0                      A0          A1

struct Window *OpenWindowTagList( struct NewWindow *,
                                   struct TagItem * );

Window = OpenWindowTags( NewWindow, Tag1, ... )

struct Window *OpenWindowTags( struct NewWindow *, ULONG, ... );
```

#### FUNCTION

A variation of OpenWindow() that allow direct specification of a TagItem array of extension data. Recommended over using the ExtNewWindow.Extension field.

If you omit the NewWindow (pass NULL), a set of defaults are used, and overridden by the tag items. Even without

any tag items at all, a reasonable window opens on the Workbench or default public screen.

See `OpenWindow()` for all the details.

#### INPUTS

`NewWindow` - (optional) pointer to a `NewWindow` structure.  
`TagItems` - (optional) pointer to `TagItem` array, with tag values as described under the description for `OpenWindow()`.

#### RESULT

`Window` - newly created window, per your specifications.

#### EXAMPLE

See `OpenScreenTagList()` for an example of how to create a "varargs" version of this function for convenient C language programming.

#### NOTES

#### BUGS

#### SEE ALSO

`OpenWindow()`

## 1.68 intuition.library/OpenWorkBench

#### NAME

`OpenWorkBench` -- Open the Workbench screen.

#### SYNOPSIS

```
WBScreen = OpenWorkBench()  
DO
```

```
BOOL OpenWorkBench( VOID );
```

#### FUNCTION

This routine attempts to reopen the Workbench. The actions taken are:

- general good stuff and nice things, and then return a non-null pointer to the Workbench screen.
- find that something has gone wrong, and return NULL

The return value, if not NULL, is indeed the address of the Workbench screen, although you should not use it as such. This is because the Workbench may be closed by other programs, which can invalidate the address at any time. We suggest that you regard the return value as a BOOL indication that the routine has succeeded, if you pay any attention to it at all.

#### INPUTS

None

#### RESULT

non-zero if Workbench screen opened successfully, or was already

---

```
opened
FALSE if anything went wrong and the Workbench screen isn't out there
```

## BUGS

The name of this routine is spelled wrong: it should be `OpenWorkbench`

SEE ALSO

### 1.69 intuition.library/PointInImage

## NAME \_\_\_\_\_

PointInImage -- Tests whether an image "contains" a point. (V36)

## SYNOPSIS

```
DoesContain = PointInImage( Point, Image )
D0          D0          A0
```

```

BOOL PointInImage( struct Point, struct Image * );

```

## FUNCTION

Tests whether a point is properly contained in an image. The intention of this is to provide custom gadgets a means to delegate "image mask" processing to the Image, where it belongs (superceding things like BOOLMASK). After all, a rounded rect image with a drop shadow knows more about what points are inside it than anybody else should.

For traditional Intuition Images, this routine checks if the point is in the Image box (LeftEdge/RightEdge/Width/Height).

## INPUTS

Point - Two words, X/Y packed into a LONG, with high word containing 'X'. This is what you get if you pass a Point structure (not a pointer!) using common C language parameter conventions.

Image - a pointer to a standard or custom Image data object.

NOTE: If 'Image' is NULL, this function returns TRUE.

## RESULT

DoesContain - Boolean result of the test.

### EXAMPLE

## NOTES

## BUGS

Only applies to the first image, does not follow NextImage linked list. This might be preferred.

SEE ALSO

## 1.70 intuition.library/PrintIText

### NAME

PrintIText -- Print text described by the IntuiText argument.

### SYNOPSIS

```
PrintIText( RastPort, IText, LeftOffset, TopOffset )
            A0          A1      D0          D1
```

```
VOID PrintIText( struct RastPort *, struct IntuiText *, WORD, WORD );
```

### FUNCTION

Prints the IntuiText into the specified RastPort. Sets up the RastPort as specified by the IntuiText values, then prints the text into the RastPort at the IntuiText x/y coordinates offset by the left/top arguments. Note, though, that the IntuiText structure itself may contain further text position coordinates: those coordinates and the Left/TopOffsets are added to obtain the true position of the text to be rendered.

This routine does window layer clipping as appropriate -- if you print text outside of your window, your characters will be clipped at the window's edge, providing you pass your window's (layered) RastPort.

If the NextText field of the IntuiText argument is non-NULL, the next IntuiText is rendered as well, and so on until some NextText field is NULL.

IntuiText with the ITextFont field NULL are displayed in the font of the RastPort. If the RastPort font is also NULL, the system default font, as set via the Preferences tool, will be used.

### INPUTS

RastPort = the RastPort destination of the text  
 IText = pointer to an instance of the structure IntuiText  
 LeftOffset = left offset of the IntuiText into the RastPort  
 TopOffset = top offset of the IntuiText into the RastPort

### RESULT

None

### BUGS

### SEE ALSO

## 1.71 intuition.library/PubScreenStatus

### NAME

PubScreenStatus -- Change status flags for a public screen. (V36)

### SYNOPSIS

```
ResultFlags = PubScreenStatus( Screen, StatusFlags )
D0                                A0      D0
```

```
UWORD PubScreenStatus( struct Screen *, UWORD );
```

#### FUNCTION

Changes status flags for a given public screen.

Do not apply this function to a screen if your program isn't the screen's "owner", in particular, don't call this function for the Workbench screen.

#### INPUTS

Screen = pointer to public screen

StatusFlags = values currently:

PSNF\_PRIVATE: make this screen unavailable to visitor windows

#### RESULT

Returns 0 in the lowest order bit of the return value if the screen wasn't public, or because it can not be taken private because visitors are open in it.

All other bits in the return code are reserved for future enhancement.

#### BUGS

#### SEE ALSO

OpenScreen(), Intuition V36 update documentation

## 1.72 intuition.library/QueryOverscan

#### NAME

QueryOverscan -- Inquire about a standard overscan region. (V36)

#### SYNOPSIS

```
success = QueryOverscan( DisplayID, Rect, OScanType )
D0                      A0          A1      D0
```

```
LONG QueryOverscan( ULONG, struct Rectangle *, WORD );
```

#### FUNCTION

This function fills in a rectangle with one of the system overscan dimensions, scaled appropriately for the mode of the DisplayID it is passed.

There are three types of system overscan values:

OSCAN\_TEXT: completely visible, by user preference. Used for Workbench screen and screen dimensions STDSCREENWIDTH and STDSCREENHEIGHT. Left/Top is always 0,0.

OSCAN\_STANDARD: just beyond visible bounds of monitor, by user preference. Left/Top may be negative.

OSCAN\_MAX: The largest region we can display, AND display any smaller region (see note below).

OSCAN\_VIDEO: The absolute largest region that the graphics.library can display. This region must be used as-is.

## INPUTS

DisplayID -- A 32-bit identifier for a display mode, as defined in graphics/displayinfo.h.

NOTE: If you only intend to use one of the four standard overscan dimensions as is, and open your screen to exactly the DisplayClip dimensions, you can specify one of the OSCAN\_ values in the ExtNewScreen tag SA\_StdDClip, and specify STDSCREENWIDTH and STDSCREENHEIGHT as the dimensions to more easily open up an overscanned screen (or use no NewScreen in OpenScreenTagList() and accept the default standard screen dimensions).

Rect -- pointer to a Rectangle structure which this function will fill out with its return values. Note that to convert a rectangle to a screen "Height" you do (MaxY - MinY + 1), and similarly for "Width." The rectangle may be passed directly to OpenScreen() as a DisplayClip region (SA\_DClip).

## RESULT

0 (FALSE) if the MonitorSpec your NewScreen requests does not exist. Non-zero (TRUE) if it does.

## BUGS

Change in parameter V36.A17 might cause problems for some.

## SEE ALSO

OpenScreen(), Intuition V36 update documentation

## 1.73 intuition.library/RefreshGadgets

## NAME

RefreshGadgets -- Refresh (redraw) the gadget display.

## SYNOPSIS

```
RefreshGadgets( Gadget, Window, Requester )
                A0      A1      A2
```

```
VOID RefreshGadgets( struct Gadget *, struct Window *,
                    struct Requester * );
```

## FUNCTION

Refreshes (redraws) all of the gadgets in the gadget list starting from the specified gadget.

The window parameter must point to the window which contains the gadget, or which contains the requester that contains the gadget. The requester parameter must only be valid if the gadget has the GTYP\_REQGADGET flag set, a requirement for all requester gadgets.

The Pointer argument points to a Window structure.

The two main reasons why you might want to use this routine are: first, that you've modified the imagery of the gadgets in your display and you want the new imagery to be displayed; secondly,

if you think that some graphic operation you just performed trashed the gadgetry of your display, this routine will refresh the imagery for you.

Note that to modify the imagery of a gadget, you must first remove that gadget from the window's gadget list, using `RemoveGadget()` (or `RemoveGList()`). After changing the image, border, text (including text for a string gadget), the gadget is replaced in the gadget list (using `AddGadget()` or `AddGList()`). Adding gadgets does not cause them to be displayed (refreshed), so this function, or `RefreshGList()` is typically called.

A common technique is to set or reset the `GFLG_SELECTED` flag of a Boolean gadget and then call `RefreshGadgets()` to see it displayed highlighted if and only if `GFLG_SELECTED` is set. If you wish to do this and be completely proper, you must `RemoveGadget()`, change the `GFLG_SELECTED` flag, `AddGadget()`, and `RefreshGadgets()`, or the equivalent.

The `gadgets` argument can be a copy of the `FirstGadget` variable in the `Window` structure that you want refreshed: the effect of this will be that all gadgets will be redrawn. However, you can selectively refresh just some of the gadgets by starting the refresh part-way into the list: for instance, redrawing your window non-`GTYP_GZZGADGET` gadgets only, which you've conveniently grouped at the end of your gadget list.

Even more control is available using the `RefreshGList()` routine which enables you to refresh a single gadget, or number of your choice.

NOTE: It's never safe to tinker with the gadget list yourself. Don't supply some gadget list that Intuition hasn't already processed in the usual way.

#### INPUTS

`Gadgets` = pointer to the first in the list of gadgets wanting refreshment  
`Window` = pointer to the window containing the gadget or its requester  
`Requester` = pointer to a requester (ignored if gadget is not attached to a requester).

#### RESULT

None

#### BUGS

#### SEE ALSO

`RefreshGList()`, `RemoveGadget()`, `RemoveGList()`, `AddGadget()`, `AddGList()`

## 1.74 intuition.library/RefreshGList

#### NAME

`RefreshGList` -- Refresh (redraw) a chosen number of gadgets.

#### SYNOPSIS

---



```
RefreshGList( Gadgets, Window, Requester, NumGad )
              A0         A1         A2         D0
```

```
VOID RefreshGList( struct Gadget *, struct Window *,
                  struct Requester *, WORD );
```

#### FUNCTION

Refreshes (redraws) gadgets in the gadget list starting from the specified gadget. At most NumGad gadgets are redrawn. If NumGad is -1, all gadgets until a terminating NULL value in the NextGadget field is found will be refreshed, making this routine a superset of RefreshGadgets().

The Requester parameter can point to a Requester structure. If the first gadget in the list has the GTYP\_REQGADGET flag set, the gadget list refers to gadgets in a requester and the pointer must necessarily point to a window. If these are not the gadgets of a requester, the requester argument may be NULL.

Be sure to see the RefreshGadgets() function description, as this function is simply an extension of that.

#### INPUTS

Gadgets = pointer to the first in the list of gadgets wanting refreshment  
 Window = pointer to the window containing the gadget or its requester  
 Requester = pointer to a requester (ignored if Gadget is not attached to a Requester).  
 NumGad = maximum number of gadgets to be refreshed. A value of -1 will cause all gadgets to be refreshed from gadget to the end of the list. A value of -2 will also do this, but if 'Gadgets' points to a Requester Gadget (GTYP\_REQGADGET) ALL gadgets in the requester will be refreshed (this is a mode compatible with v1.1 RefreshGadgets().)

#### RESULT

None

#### BUGS

#### SEE ALSO

RefreshGadgets()

## 1.75 intuition.library/RefreshWindowFrame

#### NAME

RefreshWindowFrame -- Ask Intuition to redraw your window border.

#### SYNOPSIS

```
RefreshWindowFrame( Window )
                  A0
```

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

#### FUNCTION

---

Refreshes the border of a window, including title region and all of the window's gadgets.

You may use this call if you wish to update the display of your borders. The expected use of this is to correct unavoidable corruption.

#### INPUTS

Window = a pointer to a Window structure

#### RESULT

None

#### BUGS

#### SEE ALSO

## 1.76 intuition.library/ReleaseGIRPort

#### NAME

ReleaseGIRPort -- Release a custom gadget RastPort. (V36)

#### SYNOPSIS

```
ReleaseGIRPort( RPort )
                A0
```

```
VOID ReleaseGIRPort( struct RastPort * );
```

#### FUNCTION

The corresponding function to ObtainGIRPort(), it releases arbitration used by Intuition for gadget RastPorts.

#### INPUTS

Pointer to the RastPort returned by ObtainGIRPort().  
This pointer can be NULL, in which case nothing happens.

#### RESULT

None

#### BUGS

#### SEE ALSO

ObtainGIRPort(), Custom Gadget documentation

## 1.77 intuition.library/RemakeDisplay

#### NAME

RemakeDisplay -- Remake the entire Intuition display.

#### SYNOPSIS

```
RemakeDisplay()
```

---

```
VOID RemakeDisplay( VOID );
```

**FUNCTION**

This is the big one.

This procedure remakes the entire View structure for the Intuition display. It does the equivalent of MakeScreen() for every screen in the system, and then it calls the internal equivalent of RethinkDisplay().

**WARNING:** This routine can take many milliseconds to run, so do not use it lightly.

Calling MakeScreen() followed by RethinkDisplay() is typically a more efficient method for affecting changes to a single screen's ViewPort.

**INPUTS**

None

**RESULT**

None

**BUGS****SEE ALSO**

MakeScreen(), RethinkDisplay(), graphics.library/MakeVPort()  
graphics.library/MrgCop(), graphics.library/LoadView()

## 1.78 intuition.library/RemoveClass

**NAME**

RemoveClass -- Make a public boopsi class unavailable. (V36)

**SYNOPSIS**

```
RemoveClass( classPtr )  
            A0
```

```
VOID RemoveClass( struct IClass * );
```

**FUNCTION**

Makes a public class unavailable for public consumption. It's OK to call this function for a class which is not yet in the internal public class list, or has been already removed.

**INPUTS**

ClassPtr = pointer to \*public\* class created by MakeClass(),  
may be NULL.

**RESULT**

None.

**NOTES**

BUGS

SEE ALSO

MakeClass(), FreeClass(), AddClass()  
Document "Basic Object-Oriented Programming System for Intuition"  
and the "boopsi Class Reference" document.

## 1.79 intuition.library/RemoveGadget

NAME

RemoveGadget -- Remove a gadget from a window.

SYNOPSIS

Position = RemoveGadget( Window, Gadget )  
D0                                   A0           A1

UWORD RemoveGadget( struct Window \*, struct Gadget \* );

FUNCTION

Removes the given gadget from the gadget list of the specified window. Returns the ordinal position of the removed gadget.

If the gadget is in a requester attached the the window, this routine will look for it and remove it if it is found.

If the gadget pointer points to a gadget that isn't in the appropriate list, -1 is returned. If there aren't any gadgets in the list, -1 is returned. If you remove the 65535th gadget from the list -1 is returned.

NOTES

New with V37: If one of the gadgets you wish to remove is the active gadget, this routine will wait for the user to release the mouse button before deactivating and removing the gadget.

INPUTS

Window = pointer to the window containing the gadget or the requester containing the gadget to be removed.

Gadget = pointer to the gadget to be removed. The gadget itself describes whether this is a gadget that should be removed from the window or some requester.

RESULT

Returns the ordinal position of the removed gadget. If the gadget wasn't found in the appropriate list, or if there are no gadgets in the list, returns -1.

BUGS

SEE ALSO

AddGadget(), AddGList(), RemoveGList()

## 1.80 intuition.library/RemoveGList

### NAME

RemoveGList -- Remove a sublist of gadgets from a window.

### SYNOPSIS

```
Position = RemoveGList( Window, Gadget, Numgad )
D0                      A0      A1      D0
```

```
UWORD RemoveGList( struct Window *, struct Gadget *, WORD );
```

### FUNCTION

Removes 'Numgad' gadgets from the gadget list of the specified window. Will remove gadgets from a requester if the first gadget's GadgetType flag GTYP\_REQGADGET is set.

Otherwise identical to RemoveGadget().

### NOTE

The last gadget in the list does NOT have its link zeroed.  
New with V36: OK, last gadget's NextGadget field is set to NULL.

New with V37: If one of the gadgets you wish to remove is the active gadget, this routine will wait for the user to release the mouse button before deactivating and removing the gadget.

### INPUTS

Window = pointer to the window containing the gadget or the requester containing the gadget to be removed.

Gadget = pointer to the gadget to be removed. The gadget itself describes whether this is a gadget that should be removed from the window or some requester.

Numgad = number of gadgets to be removed. If -1, remove all gadgets to end of window gadget list

### RESULT

Returns the ordinal position of the removed gadget. If the gadget wasn't found in the appropriate list, or if there are no gadgets in the list, returns -1.

### BUGS

### SEE ALSO

RemoveGadget(), AddGadget(), AddGList()

## 1.81 intuition.library/ReportMouse

### NAME

ReportMouse -- Tell Intuition whether to report mouse movement.

### SYNOPSIS

```
ReportMouse( Boolean, Window )
D0          A0          <-note
```

```
VOID ReportMouse( BOOL, struct Window * );
```

#### SPECIAL NOTE

Some compilers and link files switch the arguments to this function about in unpredictable ways. We apologize for this confusion wrapped around an error enclosing a mistake. The call will take one of two forms:

```
ReportMouse(Boolean, Window);  
-or-  
ReportMouse(Window, (ULONG) Boolean);
```

The first form is the one that corresponds to the `amiga.lib` supplied by Commodore. The linker libraries and "pragmas" of some compilers supply the alternate form.

A key point to remember is that the form of the function in ROM has always been the same, so there has always been object code compatibility. However some care should be taken when switching compilers or switching between stubs and pragmas.

From assembler the interface has always been:

```
Boolean in D0, Window in A0
```

Also, it is still endorsed to simply set the `WFLG_REPORTMOUSE` flag bit in `Window->Flags`, or reset it, on your own. Make the operation an atomic assembly instruction (`OR.W #WFLG_REPORTMOUSE, wd_Flags+2(A0)` where `A0` contains your window pointer). Most compilers will produce an atomic operation when faced with:

```
Window->Flags |= WFLG_REPORTMOUSE;  
Window->Flags &=~WFLG_REPORTMOUSE;
```

or else bracket the operation between `Forbid()/Permit()`.

#### FUNCTION

Tells Intuition whether or not to broadcast mouse-movement events to your window when it's the active one. The Boolean value specifies whether to start or stop broadcasting position information of mouse-movement. If the window is the active one, mouse-movement reports start coming immediately afterwards. This same routine will change the current state of the `GACT_FOLLOWMOUSE` function of a currently-selected gadget too.

Note that calling `ReportMouse()` when a gadget is selected will only temporarily change whether or not mouse movements are reported while that gadget remains selected; the next time the gadget is selected, its `GACT_FOLLOWMOUSE` flag is examined anew.

Note also that calling `ReportMouse()` when no gadget is currently selected will change the state of the window's `WFLG_REPORTMOUSE` flag, but will have no effect on any gadget that may be subsequently selected. (This is all fixed in V36.)

The `ReportMouse()` function is first performed when `OpenWindow()` is first called; if the flag `WFLG_REPORTMOUSE` is included among the options, then all mouse-movement events are reported to the opening task and will continue to be reported

until ReportMouse() is called with a Boolean value of FALSE.  
 If WFLG\_REPORTMOUSE is not set, then no mouse-movement reports will be broadcast until ReportMouse() is called with a Boolean of TRUE.

Note that the WFLG\_REPORTMOUSE flag, as managed by this routine, determines IF mouse messages are to be broadcast. Determining HOW they are to be broadcast is determined by the IDCMP\_MOUSEMOVE IDCMPFlag.

#### INPUTS

Window = pointer to a Window structure associated with this request  
 Boolean = TRUE or FALSE value specifying whether to turn this function on or off

#### RESULT

None

#### BUGS

See above

#### SEE ALSO

The Input and Output section of the Intuition Reference Manual

## 1.82 intuition.library/Request

#### NAME

Request -- Activate a requester.

#### SYNOPSIS

```
Success = Request( Requester, Window )
D0                A0                A1
```

```
BOOL Request( struct Requester *, struct Window * );
```

#### FUNCTION

Links in and displays a requester into the specified window.

This routine ignores the window's IDCMP\_REQVERIFY flag.

#### INPUTS

Requester = pointer to the requester to be displayed  
 Window = pointer to the window into which this requester goes

New for V36: the POINTREL flag now has meaning if the requester is not a DMR (Double-Menu Requester):  
 If POINTREL is set this requester should be placed in the center of the window, offset by Requester.RelLeft and Requester.RelTop.  
 If the requester doesn't fit in the window, its position will be adjusted to show the upper-left corner.

#### RESULT

If the requester is successfully opened, TRUE is returned. Otherwise, if the requester could not be opened, FALSE is returned.

#### BUGS

There is a maximum of 8 requesters that are supported in a window that can be changed in size, position, or depth.

SEE ALSO

The Requesters section of the Intuition Reference Manual

## 1.83 intuition.library/ResetMenuStrip

NAME

ResetMenuStrip -- Re-attach a menu strip to a window. (V36)

SYNOPSIS

```
Success = ResetMenuStrip( Window, Menu )
D0                      A0      A1
```

```
BOOL ResetMenuStrip( struct Window *, struct Menu * );
```

FUNCTION

This function is simply a "fast" version of SetMenuStrip() that doesn't perform the precalculations of menu page sizes that SetMenuStrip() does.

You may call this function ONLY IF the menu strip and all items and sub-items have not changed since the menu strip was passed to SetMenuStrip(), with the following exceptions:

- You may change the CHECKED flag to turn a checkmark on or off.
- You may change the ITEMENABLED flag to enable/disable some MenuItem or Menu structures.

In all other ways, this function performs like SetMenuStrip().

The new sequence of events you can use is:

- OpenWindow()
- SetMenuStrip()
- zero or more iterations of:
  - ClearMenuStrip()
  - change CHECKED or ITEMENABLED flags
  - ResetMenuStrip()
- ClearMenuStrip()
- CloseWindow()

INPUTS

Window = pointer to a Window structure  
Menu = pointer to the first menu in the menu strip

RESULT

TRUE always.

BUGS

SEE ALSO

SetMenuStrip(), ClearMenuStrip()

---



## 1.84 intuition.library/RethinkDisplay

### NAME

RethinkDisplay -- Grand manipulation of the entire Intuition display.

### SYNOPSIS

```
RethinkDisplay()
```

```
VOID RethinkDisplay( VOID );
```

### FUNCTION

This function performs the Intuition global display reconstruction. This includes rethinking about all of the ViewPorts and their relationship to one another and reconstructing the entire display based on the results of this rethinking.

Specifically, and omitting many internal details, the operation consists of this:

Determine which ViewPorts are invisible and set their VP\_HIDE ViewPort Mode flag. VP\_HIDE flags are also set for screens that may not be simultaneously displayed with the frontmost (V36).

If a change to a viewport height, or changing interlace or monitor scan rates require, MakeVPort() is called for specific screen viewports. After this phase, the intermediate Copper lists for each screen's viewport are correctly set up.

MrgCop() and LoadView() are then called to get these Copper lists in action, thus establishing the new state of the Intuition display.

You may perform a MakeScreen() on your Custom Screen before calling this routine. The results will be incorporated in the new display, but changing the INTERLACE ViewPort mode for one screens must be reflected in the Intuition View, which is left to Intuition.

**WARNING:** This routine can take several milliseconds to run, so do not use it lightly.

New for V36: This routine is substantially changed to support new screen modes. In particular, if screen rearrangement has caused a change in interlace mode or scan rate, this routine will remake the copper lists for each screen's viewport.

### INPUTS

None

### RESULT

None

### BUGS

In V35 and earlier, an interlaced screen coming to the front may not trigger a complete remake as required when the global interlace state is changed. In some cases, this can be compensated for by setting the viewport DHeight field to 0 for hidden screens.

SEE ALSO

```
RemakeDisplay(), graphics.library/MakeVPort(),  
graphics.library/MrgCop(), graphics.library/LoadView(), MakeScreen()
```

## 1.85 intuition.library/ScreenToBack

NAME

ScreenToBack -- Send the specified screen to the back of the display.

SYNOPSIS

```
ScreenToBack( Screen )  
            A0
```

```
VOID ScreenToBack( struct Screen * );
```

FUNCTION

Sends the specified screen to the back of the display.

INPUTS

Screen = pointer to a Screen structure

RESULT

None

BUGS

SEE ALSO

ScreenToFront()

## 1.86 intuition.library/ScreenToFront

NAME

ScreenToFront -- Make the specified screen the frontmost.

SYNOPSIS

```
ScreenToFront( Screen )  
            A0
```

```
VOID ScreenToFront( struct Screen * );
```

FUNCTION

Brings the specified Screen to the front of the display.

INPUTS

Screen = a pointer to a Screen structure

RESULT

None

BUGS

---

SEE ALSO  
ScreenToBack()

## 1.87 intuition.library/SetAttrsA

### NAME

SetAttrsA -- Specify attribute values for an object. (V36)  
SetAttrs -- Varargs stub for SetAttrsA(). (V36)

### SYNOPSIS

```
result = SetAttrsA( Object, TagList )
D0                      A0          A1

ULONG SetAttrsA( APTR, struct TagItem * );

result = SetAttrs( Object, Tag1, ... )

ULONG SetAttrs( APTR, ULONG, ... );
```

### FUNCTION

Specifies a set of attribute/value pairs with meaning as defined by a 'boopsi' object's class.

This function does not provide enough context information or arbitration for boopsi gadgets which are attached to windows or requesters. For those objects, use SetGadgetAttrs().

### INPUTS

Object = abstract pointer to a boopsi object.  
TagList = array of TagItem structures with attribute/value pairs.

### RESULT

The object does whatever it wants with the attributes you provide. The return value tends to be non-zero if the changes would require refreshing gadget imagery, if the object is a gadget.

### NOTES

This function invokes the OM\_SET method with a NULL GadgetInfo parameter.

### BUGS

### SEE ALSO

NewObject(), DisposeObject(), GetAttr(), MakeClass(),  
Document "Basic Object-Oriented Programming System for Intuition"  
and the "boopsi Class Reference" document.

## 1.88 intuition.library/SetDefaultPubScreen

### NAME

SetDefaultPubScreen -- Choose a new default public screen. (V36)

---

## SYNOPSIS

```
SetDefaultPubScreen( Name )
                    A0
```

```
VOID SetDefaultPubScreen( UBYTE * );
```

## FUNCTION

Establishes a new default public screen for visitor windows.

This screen is used by windows asking for a named public screen that doesn't exist and the FALLBACK option is selected, and for windows asking for the default public screen directly.

## INPUTS

Name = name of chosen public screen to be the new default.  
A value of NULL means that the Workbench screen is to be the default public screen.

## RESULT

None

## BUGS

## SEE ALSO

OpenWindow(), OpenScreen(), Intuition V36 update documentation

## 1.89 intuition.library/SetDMRequest

## NAME

SetDMRequest -- Set the DMRequest of a window.

## SYNOPSIS

```
success = SetDMRequest( Window, DMRequest )
D0                                A0      A1
```

```
BOOL SetDMRequest( struct Window *, struct Requester * );
```

## FUNCTION

Attempts to set the DMRequest into the specified window. The DMRequest is the special requester that you attach to the double-click of the menu button which the user can then bring up on demand. This routine WILL NOT change the DMRequest if it's already set and is currently active (in use by the user). After having called SetDMRequest(), if you want to change the DMRequest, the correct way to start is by calling ClearDMRequest() until it returns a value of TRUE; then you can call SetDMRequest() with the new DMRequest.

If the POINTREL flag is set in the DMRequest, the DMR will open as close to the pointer as possible. The RelLeft/Top fields are for fine-tuning the position.

## INPUTS

Window = pointer to the window from which the DMRequest is to be set  
DMRequest = a pointer to a requester

---

## RESULT

If the current DMRequest was not in use, sets the DMRequest pointer into the window and returns TRUE.  
If the DMRequest was currently in use, doesn't change the pointer and returns FALSE

## BUGS

## SEE ALSO

ClearDMRequest(), Request()

## 1.90 intuition.library/SetEditHook

## NAME

SetEditHook -- Set global processing for string gadgets. (V36)

## SYNOPSIS

```
OldHook = SetEditHook( Hook )
D0                                     A0

struct Hook *SetEditHook( struct Hook * );
```

## FUNCTION

Sets new global editing hook for string gadgets.

**WARNING:** This function is wholly untested. Do *\*NOT\** use this in a commercial product until further notice.

## INPUTS

Hook -- A pointer to a struct Hook which determines a function in your code to be called every time the user types a key. This is done before control is passed to the gadget custom editing hook, so effects ALL string gadgets.

## RESULT

Returns previous global edit hook structure.

## BUGS

Unknown, risky.

## SEE ALSO

## 1.91 intuition.library/SetGadgetAttrsA

## NAME

SetGadgetAttrsA -- Specify attribute values for a boopsi gadget. (V36)  
SetGadgetAttrs -- Varargs stub for SetGadgetAttrsA(). (V36)

## SYNOPSIS

```
result = SetGadgetAttrsA( Gadget, Window, Requester, TagList )
D0                                     A0      A1      A2      A3
```

```

ULONG SetGadgetAttrs( struct Gadget *, struct Window *,
                      struct Requester *, struct TagItem * );

result = SetGadgetAttrs( Gadget, Window, Requester, Tag1, ...)

ULONG SetGadgetAttrs( struct Gadget *, struct Window *,
                      struct Requester *, ULONG, ... );

```

#### FUNCTION

Same as `SetAttrs()`, but provides context information and arbitration for classes which implement custom Intuition gadgets.

You should use this function for boopsi gadget objects which have already been added to a requester or a window, or for "models" which propagate information to gadget already added.

Typically, the gadgets will refresh their visuals to reflect changes to visible attributes, such as the value of a slider, the text in a string-type gadget, the selected state of a button.

You can use this as a replacement for `SetAttrs()`, too, if you specify `NULL` for the 'Window' and 'Requester' parameters.

#### INPUTS

Gadget = abstract pointer to a boopsi gadget  
 Window = window gadget has been added to using `AddGLList()` or `AddGadget()`  
 Requester = for `REQGADGETs`, requester containing the gadget  
 TagList = array of `TagItem` structures with attribute/value pairs.

#### RESULT

The object does whatever it wants with the attributes you provide, which might include updating its gadget visuals.

The return value tends to be non-zero if the changes would require refreshing gadget imagery, if the object is a gadget.

#### NOTES

This function invokes the `OM_SET` method with a `GadgetInfo` derived from the 'Window' and 'Requester' pointers.

#### BUGS

There should be more arbitration between this function and the calls that Intuition's input task will make to the gadgets. In the meantime, this function, input processing, and refreshing must be mutually re-entrant.

#### SEE ALSO

`NewObject()`, `DisposeObject()`, `GetAttr()`, `MakeClass()`,  
 Document "Basic Object-Oriented Programming System for Intuition"  
 and the "boopsi Class Reference" document.

## 1.92 intuition.library/SetMenuStrip

## NAME

SetMenuStrip -- Attach a menu strip to a window.

## SYNOPSIS

```
Success = SetMenuStrip( Window, Menu )
D0                      A0      A1
```

```
BOOL SetMenuStrip( struct Window *, struct Menu * );
```

## FUNCTION

Attaches the menu strip to the window. After calling this routine, if the user presses the menu button, this specified menu strip will be displayed and accessible by the user.

Menus with zero menu items are not allowed.

NOTE: You should always design your menu strip changes to be a two-way operation, where for every menu strip you add to your window you should always plan to clear that strip sometime. Even in the simplest case, where you will have just one menu strip for the lifetime of your window, you should always clear the menu strip before closing the window. If you already have a menu strip attached to this window, the correct procedure for changing to a new menu strip involves calling `ClearMenuStrip()` to clear the old first.

The sequence of events should be:

- `OpenWindow()`
- zero or more iterations of:
  - `SetMenuStrip()`
  - `ClearMenuStrip()`
- `CloseWindow()`

## INPUTS

Window = pointer to a Window structure

Menu = pointer to the first menu in the menu strip

## RESULT

TRUE if there were no problems. TRUE always, since this routine will wait until it is OK to proceed.

## BUGS

## SEE ALSO

`ClearMenuStrip()`, `ResetMenuStrip()`

## 1.93 intuition.library/SetMouseQueue

## NAME

SetMouseQueue -- Change limit on pending mouse messages. (V36)

## SYNOPSIS

```
oldQueueLength = SetMouseQueue( Window, QueueLength )
D0                      A0      D0
```

```
LONG SetMouseQueue( struct Window *, UWORD );
```

#### FUNCTION

Changes the number of mouse messages that Intuition will allow to be outstanding for your window.

#### INPUTS

Window = your window

QueueLength = the new value of outstanding mouse movement messages you wish to allow.

#### RESULT

-1 if 'Window' is not known

Otherwise the previous value of the queue limit.

The corresponding function for changing the repeat key queue limit is not yet implemented.

#### BUGS

#### SEE ALSO

OpenWindow()

## 1.94 intuition.library/SetPointer

#### NAME

SetPointer -- Specify a pointer sprite image for a window.

#### SYNOPSIS

```
SetPointer( Window, Pointer, Height, Width, XOffset, YOffset )
           A0      A1      D0      D1      D2      D3
```

```
VOID SetPointer( struct Window *, UWORD *, WORD, WORD, WORD, WORD );
```

#### FUNCTION

Sets up the window with the sprite definition for the pointer. Then, whenever the window is the active one, the pointer image will change to the window's version. If the window is the active one when this routine is called, the change takes place immediately.

The XOffset and YOffset parameters are used to offset the upper-left corner of the hardware sprite image from what Intuition regards as the current position of the pointer. Another way of describing it is as the offset from the "hot spot" of the pointer to the top-left corner of the sprite. For instance, if you specify offsets of zero, zero, then the top-left corner of your sprite image will be placed at the mouse position. On the other hand, if you specify an XOffset of -7 (remember, sprites are 16 pixels wide) then your sprite will be centered over the mouse position. If you specify an XOffset of -15, the right-edge of the sprite will be over the mouse position.

#### INPUTS

Window = pointer to the window to receive this pointer definition

Pointer = pointer to the data definition of a sprite

---



Height = the height of the pointer  
Width = the width of the sprite (must be less than or equal to sixteen)  
XOffset = the offset for your sprite from the mouse position  
YOffset = the offset for your sprite from the mouse position

**RESULT**

None

**BUGS****SEE ALSO**

ClearPointer(), Intuition Reference Manual or Amiga Rom  
Kernel Manual

## 1.95 intuition.library/SetPrefs

**NAME**

SetPrefs -- Set Intuition preferences data.

**SYNOPSIS**

```
Prefs = SetPrefs( PrefBuffer, Size, Inform )  
D0                      A0                      D0      D1
```

```
struct Preferences *SetPrefs( struct Preferences *, LONG, BOOL );
```

**FUNCTION**

Sets new preferences values. Copies the first 'Size' bytes from your preferences buffer to the system preferences table, and puts them into effect.

The 'Inform' parameter, if TRUE, indicates that an IDCMP\_NEWPREFS message is to be sent to all windows that have the IDCMP\_NEWPREFS IDCMPFlag set.

It is legal to set a partial copy of the Preferences structure. The most frequently changed values are grouped at the beginning of the Preferences structure.

New for V36: A new and more extensible method for supplying Preferences has been introduced in V36, and relies on file system notification. The Intuition preferences items rely also on the IPrefs program. Certain elements of the Preferences structure have been superceded by this new method. Pointer changes submitted through SetPrefs() are only heeded until the first time IPrefs informs Intuition of a V36-style pointer.ilbm preferences file. The Preferences FontHeight and LaceWB fields are respected only from the system-configuration file, and never thereafter. As well, the view centering and size apply only to the default monitor, and not to such modes as Productivity. Other fields may be superceded in the future.

**INPUTS**

PrefBuffer = pointer to the memory buffer which contains your  
desired settings for Intuition preferences  
Size = the number of bytes in your PrefBuffer, the number of bytes

you want copied to the system's internal preference settings  
 Inform = whether you want the information of a new preferences  
 setting propagated to all windows.

## NOTES

Unless you are responding to a user's explicit request to change Preferences (for example, you are writing a Preferences editor), you should probably avoid using this function. The user's Preferences should be respected, not overridden.

## RESULT

Returns your parameter PrefBuffer.

## BUGS

## SEE ALSO

GetDefPrefs(), GetPrefs()

## 1.96 intuition.library/SetPubScreenModes

## NAME

SetPubScreenModes -- Establish global public screen behavior. (V36)

## SYNOPSIS

```
OldModes = SetPubScreenModes( Modes )
D0                                     D0

UWORD SetPubScreenModes( UWORD );
```

## FUNCTION

Sets GLOBAL Intuition public screen modes.

## INPUTS

Modes = new global modes flags. Values for flag bits are:  
 SHANGHAI: workbench windows are to be opened on the  
           default public screen  
 POPPUBSCREEN: when a visitor window is opened, the public  
           screen it opens on is to be brought to the front.

## RESULT

OldModes = previous global mode settings

## BUGS

## SEE ALSO

OpenScreen(), Intuition V36 update documentation

## 1.97 intuition.library/SetWindowTitles

## NAME

SetWindowTitles -- Set the window's titles for both window and screen.

---

## SYNOPSIS

```
SetWindowTitles( Window, WindowTitle, ScreenTitle )
                  A0      A1          A2
```

```
VOID SetWindowTitles( struct Window *, UBYTE *, UBYTE * );
```

## FUNCTION

Allows you to set the text which appears in the Window and/or Screen title bars.

The window title appears at all times along the window title bar. The window's screen title appears at the screen title bar whenever this window is the active one.

When this routine is called, your window title will be changed immediately. If your window is the active one when this routine is called, the screen title will be changed immediately.

You can specify a value of -1 (i.e. (UBYTE \*) ~0) for either of the title pointers. This designates that you want Intuition to leave the current setting of that particular title alone, and modify only the other one. Of course, you could set both to -1.

Furthermore, you can set a value of 0 (zero) for either of the title pointers. Doing so specifies that you want no title to appear (the title bar will be blank).

Both of the titles are rendered in the default font of the window's screen, as set using `OpenScreen()`.

In setting the window's title, Intuition may do some other rendering in the top border of your window. If your own rendering sometimes appears in your window border areas, you may want to restore the entire window border frame. The function `SetWindowTitles()` does not do this in the newer versions. The function `RefreshWindowFrame()` is provided to do this kind of thing for you.

## INPUTS

Window = pointer to your window structure  
WindowTitle = pointer to a null-terminated text string, or set to either the value of -1 (negative one) or 0 (zero)  
ScreenTitle = pointer to a null-terminated text string, or set to either the value of -1 (negative one) or 0 (zero)

## RESULT

None

## BUGS

## SEE ALSO

`OpenWindow()`, `RefreshWindowFrame()`, `OpenScreen()`

## 1.98 intuition.library/ShowTitle

---

## NAME

ShowTitle -- Set the screen title bar display mode.

## SYNOPSIS

```
ShowTitle( Screen, ShowIt )
           A0      D0
```

```
VOID ShowTitle( struct Screen *, BOOL );
```

## FUNCTION

This routine sets the SHOWTITLE flag of the specified screen, and then coordinates the redisplay of the screen and its windows.

The screen title bar can appear either in front of or behind WFLG\_BACKDROP windows. This is contrasted with the fact that non-WFLG\_BACKDROP windows always appear in front of the screen title bar. You specify whether you want the screen title bar to be in front of or behind the screen's WFLG\_BACKDROP windows by calling this routine.

The ShowIt argument should be set to either TRUE or FALSE. If TRUE, the screen's title bar will be shown in front of WFLG\_BACKDROP windows

If FALSE, the title bar will be rendered behind all windows.

When a screen is first opened, the default setting of the SHOWTITLE flag is TRUE.

## INPUTS

Screen = pointer to a Screen structure  
ShowIt = Boolean TRUE or FALSE describing whether to show or hide the screen title bar

## RESULT

None

## BUGS

## SEE ALSO

## 1.99 intuition.library/SizeWindow

## NAME

SizeWindow -- Ask Intuition to size a window.

## SYNOPSIS

```
SizeWindow( Window, DeltaX, DeltaY )
           A0      D0      D1
```

```
VOID SizeWindow( struct Window *, WORD, WORD );
```

## FUNCTION

This routine sends a request to Intuition asking to size the window the specified amounts. The delta arguments describe how much to

---

size the window along the respective axes.

Note that the window will not be sized immediately, but rather will be sized the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second, and a maximum of sixty times a second. You can discover when your window has finally been sized by setting the IDCMP\_NEWSIZE flag of the IDCMP of your window. See the "Input and Output Methods" chapter of The Intuition Reference Manual for description of the IDCMP.

New for V36: Intuition now will do validity checking on the final dimensions. To change to new absolute dimensions, or to move and size a window in one step, use `ChangeWindowBox()`.

However, limit checking against window `MinWidth`, `MinHeight`, `MaxWidth`, and `MaxHeight` was not done prior to V36, and these fields are still ignored (as documented) if you have no sizing gadget (`WFLG_SIZEGADGET` is not set). The `*are*` respected now (V36) if `WFLG_SIZEGADGET` is set.

New for V36: you can determine when the change in size has taken effect by receiving the `IDCMP_CHANGEWINDOW` IDCMP message.

#### INPUTS

`Window` = pointer to the structure of the window to be sized  
`DeltaX` = signed value describing how much to size the window on the x-axis  
`DeltaY` = signed value describing how much to size the window on the y-axis

#### RESULT

None

#### BUGS

#### SEE ALSO

`ChangeWindowBox()`, `MoveWindow()`, `WindowToFront()`, `WindowToBack()`

## 1.100 intuition.library/SysReqHandler

#### NAME

`SysReqHandler` -- Handle system requester input. (V36)

#### SYNOPSIS

```
num = SysReqHandler( Window, IDCMPFlagsPtr, WaitInput )
D0                      A0          A1          D0
```

```
LONG SysReqHandler( struct Window *, ULONG *, BOOL );
```

#### FUNCTION

Handles input for a window returned by either `BuildSysRequest()` or `BuildEasyRequest()`. These functions with `SysReqHandler()` you can perform an "asynchronous" `EasyRequest()` or `AutoRequest()`. That is to say, you can perform other processing while you wait for the requester to be satisfied.

Each time this function is called, it will process all IDCMPMessages that the window has received. If the parameter 'WaitInput' is non-zero, SysReqHandler() will wait for input (by calling WaitPort()) if there are no IDCMP messages.

SysReqHandler() returns the same values as EasyRequest(): A gadget ID greater than equal to 0, and -1 if one of the other IDCMP events were received.

An additional value of -2 is returned if the input processed does not satisfy the requester. In this case, you might perform some processing and call SysReqHandler() again.

Note: this function does NOT terminate the system request. Not only must you call FreeSysRequest() to eliminate the request, but you may also continue processing after an event which would normally terminate a normal call to EasyRequest().

#### EXAMPLE

Implementation of EasyRequest() input loop:

```
window = BuildEasyRequest( ZZZZ )
while ( (retval = SysReqHandler( window, idcmp_ptr, TRUE )) == -2 )
{
    /* loop      */
}
FreeSysRequest( window );
```

#### EXAMPLE

Request a volume, but don't remove the requester when the user inserts the wrong disk:

```
struct EasyStruct volumeES = {
    sizeof (struct EasyStruct),
    0,
    "Volume Request",
    "Please insert volume %s in any drive.",
    "Cancel"
};

Volume *
getVolume( volname )
UBYTE  *volname;
{
    struct Window *window;
    Volume      *volume = NULL;
    Volume      *findVolume();
    int         retval;

    window = BuildEasyRequest( NULL, &volumeES, IDCMP_DISKINSERTED,
                               volname );

    while ( (retval = SysReqHandler( window, NULL, TRUE )) != 0 )
    {
        /* not cancelled yet      */
    }
```

```

        /* when IDCMP_DISKINSERTED, check for volume */
        if (( retval == -1 ) && (volume = findVolume( volname )))
            break;
    }
    FreeSysRequest( window );
    return ( volume );
}

```

#### INPUTS

Window = Window pointer returned from BuildSysRequest() or BuildEasyRequest(). Those functions can also return values '0' or '1', and these values may also be passed to SysReqHandler(), which will immediately return the same value.

IDCMPFlagsPtr = If you passed application specific IDCMP flags to BuildSysRequest() or BuildEasyRequest(), SysReqHandler() will return -1 if that IDCMP message is received. If IDCMPFlagsPtr is non-null, it points to a ULONG where the IDCMP class received will be copied for your examination.

This pointer can be NULL if you have provided no application specific IDCMP flags or if you do not need to know which application specific IDCMP event occurred.

If you provide more than one flag in the flags variable this pointer points to, you will have to refresh the variable whenever -1 is returned, since the variable will have been changed to show just the single IDCMP Class bit that caused the return.

WaitInput = Specifies that you want SysReqHandler() to wait for IDCMP input if there is none pending.

#### RESULT

0, 1, ..., N = Successive GadgetID values, for the gadgets you specify for the requester. NOTE: The numbering from left to right is actually: 1, 2, ..., N, 0. This is for compatibility with AutoRequests which has FALSE for the rightmost gadget.

-1 = Means that one of the caller-supplied IDCMPFlags occurred. The IDCMPFlag value is in the longword pointed to by IDCMP\_ptr.

-2 = input processed did not satisfy the requester. One example is a keystroke that does not satisfy the requester. Another example is if there is no input pending and you specified FALSE for WaitInput.

#### BUGS

#### SEE ALSO

exec.library/WaitPort()

## 1.101 intuition.library/UnlockIBase

### NAME

UnlockIBase -- Surrender an Intuition lock gotten by LockIBase().

### SYNOPSIS

```
UnlockIBase( Lock )
            A0
```

```
VOID UnlockIBase( ULONG );
```

### FUNCTION

Surrenders lock gotten by LockIBase().

Calling this function when you do not own the specified lock will immediately crash the system.

### INPUTS

The value returned by LockIBase() should be passed to this function, to specify which internal lock is to be freed.

Note that the parameter is passed in A0, not D0, for historical reasons.

### RESULT

None

### BUGS

### SEE ALSO

LockIBase()

## 1.102 intuition.library/UnlockPubScreen

### NAME

UnlockPubScreen -- Release lock on a public screen. (V36)

### SYNOPSIS

```
UnlockPubScreen( Name, [Screen] )
                A0     A1
```

```
VOID UnlockPubScreen( UBYTE *, struct Screen * );
```

### FUNCTION

Releases lock gotten by LockPubScreen().

It is best to identify the locked public screen by the pointer returned from LockPubScreen(). To do this, supply a NULL 'Name' pointer and the screen pointer.

In rare circumstances where it would be more convenient to pass a non-NULL pointer to the public screen name string, the 'Screen' parameter is ignored.

### INPUTS

---



Name = pointer to name of public screen. If Name is NULL, then argument 'Screen' is used as a direct pointer to a public screen.  
Screen = pointer to a public screen. Used only if Name is NULL. This pointer MUST have been returned by LockPubScreen().

RESULT

BUGS

SEE ALSO

LockPubScreen()

### 1.103 intuition.library/UnlockPubScreenList

NAME

UnlockPubScreenList -- Release public screen list semaphore. (V36)

SYNOPSIS

UnlockPubScreenList()

VOID UnlockPubScreenList( VOID );

FUNCTION

Releases lock gotten by LockPubScreenList().

INPUTS

None.

RESULT

None.

BUGS

SEE ALSO

LockPubScreenList()

### 1.104 intuition.library/ViewAddress

NAME

ViewAddress -- Return the address of the Intuition View structure.

SYNOPSIS

view = ViewAddress()  
DO

struct View \*ViewAddress( VOID );

FUNCTION

Returns the address of the Intuition View structure. If you want to use any of the graphics, text, or animation primitives

in your window and that primitive requires a pointer to a view, this routine will return the address of the view for you.

**INPUTS**

None

**RESULT**

Returns the address of the Intuition View structure

**BUGS****SEE ALSO**

graphics.library

## 1.105 intuition.library/ViewPortAddress

**NAME**

ViewPortAddress -- Return the address of a window's viewport.

**SYNOPSIS**

```
ViewPort = ViewPortAddress( Window )  
D0                                A0
```

```
struct ViewPort *ViewPortAddress( struct Window * );
```

**FUNCTION**

Returns the address of the viewport associated with the specified window. The viewport is actually the viewport of the screen within which the window is displayed. If you want to use any of the graphics, text, or animation primitives in your window and that primitive requires a pointer to a viewport, you can use this call.

This pointer is only valid as long as your window's screen remains open, which is ensured by keeping your window open.

**INPUTS**

Window = pointer to the window for which you want the viewport address

**RESULT**

Returns the address of the Intuition ViewPort structure for your window's screen .

**BUGS**

This routine is unnecessary: you can just use the expression &Window->WScreen->ViewPort.

**SEE ALSO**

graphics.library

## 1.106 intuition.library/WBenchToBack

## NAME

WBenchToBack -- Send the Workbench screen in back of all screens.

## SYNOPSIS

```
Success = WBenchToBack()  
D0
```

```
BOOL WBenchToBack( VOID );
```

## FUNCTION

Causes the Workbench screen, if it's currently opened, to go behind all other screens. This does not 'move' the screen up or down, instead only affects the depth-arrangement of the screens.

## INPUTS

None

## RESULT

If the Workbench screen was opened, this function returns TRUE, otherwise it returns FALSE.

## BUGS

## SEE ALSO

WBenchToFront(), ScreenToFront()

## 1.107 intuition.library/WBenchToFront

## NAME

WBenchToFront -- Bring the Workbench screen in front of all screens.

## SYNOPSIS

```
Success = WBenchToFront()  
D0
```

```
BOOL WBenchToFront( VOID );
```

## FUNCTION

Causes the Workbench Screen, if it's currently opened, to come to the foreground. This does not 'move' the screen up or down, instead only affects the depth-arrangement of the screen.

## INPUTS

None

## RESULT

If the Workbench screen was opened, this function returns TRUE, otherwise it returns FALSE.

## BUGS

## SEE ALSO

WBenchToBack(), ScreenToBack()

---

## 1.108 intuition.library/WindowLimits

### NAME

WindowLimits -- Set the minimum and maximum limits of a window.

### SYNOPSIS

```
Success = WindowLimits( Window, MinWidth, MinHeight, MaxWidth,  
D0                      A0      D0      D1      D2  
                        MaxHeight )  
D3
```

```
BOOL WindowLimits( struct Window *, WORD, WORD, UWORD, UWORD );
```

### FUNCTION

Sets the minimum and maximum limits of the window's size. Until this routine is called, the window's size limits are equal to the initial values established in the OpenWindow() function.

After a call to this routine, the Window will be able to be sized to any dimensions within the specified limits.

If you don't want to change any one of the dimensions, set the limit argument for that dimension to zero. If any of the limit arguments is equal to zero, that argument is ignored and the initial setting of that parameter remains undisturbed.

If any of the arguments is out of range (minimums greater than the current size, maximums less than the current size), that limit will be ignored, though the others will still take effect if they are in range. If any are out of range, the return value from this procedure will be FALSE. If all arguments are valid, the return value will be TRUE.

If you want your window to be able to become "as large as possible" you may put -1 (i.e. ~0) in either or both Max arguments. But please note: screen sizes may vary for several reasons, and you must be able to handle any possible size of window you might end up with if you use this method. Note that you can use the function GetScreenData() to find out how big the screen your window appears in is. That function is particularly useful if your window is in the Workbench screen. You may also refer to the WScreen field in your window structure, providing that your window remains open, which will ensure that the screen remains open, and thus the pointer remains valid.

If the user is currently sizing this window, the new limits will not take effect until after the sizing is completed.

### INPUTS

Window = pointer to a Window structure  
MinWidth, MinHeight, MaxWidth, MaxHeight = the new limits for the size of this window. If any of these is set to zero, it will be ignored and that setting will be unchanged.

### RESULT

Returns TRUE if everything was in order. If any of the parameters was

out of range (minimums greater than current size, maximums less than current size), FALSE is returned and the errant limit request is not fulfilled (though the valid ones will be).

BUGS

SEE ALSO

GetScreenData()

## 1.109 intuition.library/WindowToBack

NAME

WindowToBack -- Ask Intuition to send a window behind others.

SYNOPSIS

```
WindowToBack( Window )
              A0
```

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

FUNCTION

This routine sends a request to Intuition asking to send the window in back of all other windows in the screen.

Note that the window will not be depth-arranged immediately, but rather will be arranged the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second, and a maximum of sixty times a second.

Remember that WFLG\_BACKDROP windows cannot be depth-arranged.

INPUTS

Window = pointer to the structure of the window to be sent to the back

RESULT

None

BUGS

SEE ALSO

MoveWindow(), SizeWindow(), WindowToFront(), MoveWindowInFrontOf()

## 1.110 intuition.library/WindowToFront

NAME

WindowToFront -- Ask Intuition to bring a window to the front.

SYNOPSIS

```
WindowToFront( Window )
              A0
```

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

---

## FUNCTION

This routine sends a request to Intuition asking to bring the window in front of all other windows in the screen.

Note that the window will not be depth-arranged immediately, but rather will be arranged the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second, and a maximum of sixty times a second.

Remember that WFLG\_BACKDROP windows cannot be depth-arranged.

## INPUTS

Window = pointer to the structure of the window to be brought to front

## RESULT

None

## BUGS

## SEE ALSO

MoveWindow(), SizeWindow(), WindowToBack(), MoveWindowInFrontOf()

## 1.111 intuition.library/ZipWindow

## NAME

ZipWindow -- Change window to "alternate" position and dimensions. (V36)

## SYNOPSIS

```
ZipWindow( Window )
          A0
```

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

## FUNCTION

Changes the position and dimension of a window to the values at the last occasion of ZipWindow being called (or invoked via the "zoom" gadget).

Typically this is used to snap between a normal, large, working dimension of the window to a smaller, more innocuous position and dimension.

Like MoveWindow(), SizeWindow(), and ChangeWindowBox(), the action of this function is deferred to the Intuition input handler.

More tuning needs to be done to establish initial values for the first invocation of this function for a window. You can provide initial values using the OpenWindow() tag item WA\_Zoom.

It could also use a new name, but "ZoomWindow" is misleading, since "Zoom" normally implies "scale."

---

The zoom gadget will appear (in the place of the old "toback" gadget) when you open your window if you either specify a sizing gadget or use `WA_Zoom`.

You can detect that this function has taken effect by receiving an `IDCMP_CHANGEWINDOW` `IDCMP` message.

#### INPUTS

Window -- window to be changed.

#### RESULT

None

#### BUGS

`OpenWindow()` assumes that the proper default "other" dimensions are "full size."

#### SEE ALSO

`ChangeWindowBox()`, `MoveWindow()`, `SizeWindow()`