# intuition

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>intuition | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | March 28, 2025 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# intuition

## 1.1 intuition.doc

## 1.2 intuition.library/ActivateGadget

```
NAME
    ActivateGadget -- Activate a (String) Gadget.

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

    BOOL   Success;
    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Request;

FUNCTION
    Activates a String 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 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. (Use the ACTIVEWINDOW IDCMP).
    -   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 the REQSET and REQCLEAR IDCMP).
    -   The right mouse button cannot be held down (e.g. menus

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 by NULL if this isn't
       a Requester Gadget (i.e. 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

SEE ALSO
```

## 1.3 intuition.library/ActivateWindow

```
NAME
    ActivateWindow  --  Activate an Intuition Window.

SYNOPSIS
    ActivateWindow(Window)
                   A0

    struct Window *Window;

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 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
    None

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

SEE ALSO
    OpenWindow(), and the ACTIVATE Window Flag
```

## 1.4 intuition.library/AddGadget

```
NAME
    AddGadget  --  Add a Gadget to the Gadget list of the Window or Screen.

SYNOPSIS
    RealPosition = AddGadget(Window, Gadget, Position)
    D0                       A0      A1      D0

    USHORT RealPosition;
    struct Window *Window;
    struct Gadget *Gadget;
    USHORT Position;

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 Pos == 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 do be redisplayed.
    The benefit of this is that you may add several gadgets without having
    the gadget list be 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., (USHORT) ~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()
```

## 1.5 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

    USHORT RealPosition;
    struct Window *Window;
    struct Gadget *Gadget;
    USHORT Position;
    USHORT Numgad;
    struct Requester *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.

    The Requester parameter will be ignored unless the 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 REQGADGET is either set of 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
        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()
```

## 1.6 intuition.library/AllocRemember

```
NAME
    AllocRemember -- AllocMem and create a link node to make FreeMem easy.

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

    CPTR MemBlock;
    struct Remember **RememberKey;
    ULONG Size;
    ULONG Flags;

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 (such as the Intuition OpenWindow() procedure).
       If any one of the allocations fails for lack of memory, you
       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;
    AllocRemember(&RememberKey, BUFSIZE, MEMF_CHIP);
    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 undisturbed.

BUGS

SEE ALSO
```

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

## 1.7 intuition.library/AutoRequest

```
NAME
    AutoRequest  --  Automatically build and get response from a Requester.

SYNOPSIS
    Response = AutoRequest(Window, BodyText, PositiveText, NegativeText,
    D0                      A0      A1        A2            A3
                    PositiveFlags, NegativeFlags, Width, Height)
                    D0             D1             D2     D3

    BOOL Response;
    struct Window *Window;
    struct IntuiText *BodyText, *PositiveText, *NegativeText;
    ULONG PositiveFlags, NegativeFlags;
    SHORT Width, Height;

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 creates by bitwise "or'ing" your
    PositiveFlags, NegativeFlags, and the IDCMP classes GADGETUP and
    RAWKEY.  You may specify zero flags for either the PositiveFlags
    or NegativeFlags 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 PositiveFlags arguments (if you've supplied
       any), in which case this routine returns TRUE.  Or
  -   the message class matches one of your NegativeFlags
       arguments (if you've supplied any), in which case
       this routine returns FALSE.  Or
  -   the IDCMP message is of class GADGETUP, which means that one of
        the two Gadgets, as provided with the PositiveText and NegativeText
       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 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.

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

INPUTS
    Window = pointer to a Window structure
    BodyText = pointer to an IntuiText structure
    PositiveText = pointer to an IntuiText structure, may be NULL.
    NegativeText = pointer to an IntuiText structure, MUST be valid!
    PositiveFlags = flags for the IDCMP
    NegativeFlags = flags for the IDCMP
    Width, Height = the sizes to be used for the rendering of the
                    Requester

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

SEE ALSO
    BuildSysRequest()
```

## 1.8 intuition.library/BeginRefresh

```
NAME
    BeginRefresh  --  Sets up a Window for optimized refreshing.

SYNOPSIS
    BeginRefresh(Window)
                 A0

    struct Window *Window;

FUNCTION
    This routine sets up your Window for optimized refreshing.

    It’s 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 GIMMEZEROZERO window only after the
    parent Layer_Info has been locked.  Also, the 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 in
    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 SIMPLE_REFRESH Window which is partially
    concealed and the user brings it to the front, you may receive a
    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 such 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 Window as usual.

    You learn that your Window needs refreshing by receiving either a
    message of class 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 NOCAREREFRESH Flag when
    opening your window.

INPUTS
    Window = pointer to the Window structure which needs refreshing

RESULT
    None

BUGS

SEE ALSO
    EndRefresh(), layers.library/BeginUpdate(), OpenWindow()
    The "Windows" chapter of the Intuition Reference Manual
```

## 1.9 intuition.library/BuildSysRequest

```
NAME
    BuildSysRequest  --  Build and display a system Requester.

SYNOPSIS
    ReqWindow = BuildSysRequest(Window, BodyText, PositiveText,
    D0                          A0      A1        A2
                      NegativeText, IDCMPFlags, Width, Height)
                      A3            D0          D2     D3

    struct Window *ReqWindow;
    struct Window *Window;
    struct IntuiText *BodyText;
    struct IntuiText *PositiveText;
    struct IntuiText *NegativeText;
    ULONG  IDCMPFlags;
    SHORT  Width, Height;

FUNCTION
```

```
This procedure builds a 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 the IDCMP UserPort and WindowPort 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 DISKINSERTED,
for instance).  After the Requester is satisfied, you should call the
FreeSysRequest() procedure to remove the Requester and free up any
allocated memory.

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 RAWKEY, MOUSEBUTTONS, and others.

If it isn't possible to construct the Requester for any reason, this
procedure will instead use the text arguments to construct a text
string for a call to the DisplayAlert() procedure, and then will return
either a TRUE or FALSE depending on whether DisplayAlert() returned
a FALSE or TRUE respectively.

If the Window argument you supply is equal to NULL, a new Window will
be created for you in the Workbench Screen.  If you want the Requester
created by this routine to be bound to a particular Window, you should
not supply a Window argument of NULL.

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 PositiveText 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 NegativeText 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:
      - BOOLGADGET
      - RELVERIFY
      - REQGADGET
      - 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.

    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 will be provided as a function distinct from this
    one.

INPUTS
    Window = pointer to a Window structure
    BodyText = pointer to an IntuiText structure
    PositiveText = pointer to an IntuiText structure
    NegativeText = 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

RESULT
    If the Requester was successfully rendered in a Window, the value
    returned by this procedure is a pointer to the Window in which the
    Requester was rendered.  If, however, the Requester cannot be rendered
    in the Window, this routine will have called DisplayAlert() before
    returning and will pass back TRUE if the user pressed the left mouse
    button and FALSE if the user pressed the right mouse button.

BUGS
    This procedure currently opens a Window as wide as the Screen in
    which it was rendered, and then opens the Requester within that
    Window.  Also, if DisplayAlert() is called, the PositiveText and
    NegativeText are not rendered in the lower corners of the Alert.

SEE ALSO
    FreeSysRequest(), DisplayAlert(), ModifyIDCMP(), exec.library/Wait(),
    Request(), AutoRequest()
```

## 1.10 intuition.library/ClearDMRequest

```
NAME
    ClearDMRequest -- clears (detaches) the DMRequest of the Window.
```

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

    BOOL Response;
    struct Window *Window;

FUNCTION
    Attempts to clear the DMRequester 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 DMRequester
    if it's active (in use by the user). The IDCMP message class 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, zeroes out the DMRequest
    pointer in the Window and returns TRUE.
pointer in the Window and returns TRUE.
    If the DMRequest was currently in use, doesn't change the pointer
    and returns FALSE.

BUGS

SEE ALSO
    SetDMRequest(), Request()
```

## 1.11 intuition.library/ClearMenuStrip

```
NAME
    ClearMenuStrip  --  Clears (detaches) the Menu strip from the Window

SYNOPSIS
    ClearMenuStrip(Window)
                   A0

    struct Window *Window;

FUNCTION
    Detaches the current menu strip from the Window; menu strips
    are attached to windows using the SetMenuStrip() function.

    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()
```

## 1.12 intuition.library/ClearPointer

```
NAME
    ClearPointer  --  clears the Mouse Pointer definition from a Window.

SYNOPSIS
    ClearPointer(Window)
                 A0

    struct Window *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()
```

## 1.13 intuition.library/CloseScreen

```
NAME
    CloseScreen  --  Closes an Intuition Screen.

SYNOPSIS
    CloseScreen(Screen)
                A0
    struct Screen *Screen;

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

INPUTS
    Screen = pointer to the Screen to be closed.

RESULT
    None

BUGS

SEE ALSO
    OpenScreen()
```

## 1.14 intuition.library/CloseWindow

```
NAME
    CloseWindow  --  Closes an Intuition Window.

SYNOPSIS
    CloseWindow(Window)
                A0

    struct Window *Window;

FUNCTION
    Closes an Intuition Window.  Unlinks it from the system, unallocates
    its memory, and if its Screen is a system one that would be empty
    without the Window, closes the system Screen too.

    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.
    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 delink and free
    your window.  Your program may thus be suspended while the user
    plays with gadgets, menus, or window sizes and position.

INPUTS
    Window = a pointer to a Window structure
```

```
RESULT
    None

BUGS

SEE ALSO
    OpenWindow(), CloseScreen()
```

## 1.15 intuition.library/CloseWorkBench

```
NAME
    CloseWorkBench  --  Closes the Workbench Screen.

SYNOPSIS
    Success =  CloseWorkBench()
    D0

    BOOL Success;

FUNCTION
    This routine attempts to close the Workbench.  The actions taken are:
   -   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.

BUGS

SEE ALSO
    OpenWindow()
```

## 1.16 intuition.library/CurrentTime

```
NAME
    CurrentTime  --  Get the current time values.

SYNOPSIS
    CurrentTime(Seconds, Micros)
                A0       A1

    ULONG *Seconds, *Micros;
```

```
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
    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
```

## 1.17 intuition.library/DisplayAlert

```
NAME
    DisplayAlert  --  Create the display of an Alert message.

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

    BOOL  Response;
    ULONG AlertNumber;
    UBYTE *String;
    SHORT Height;

FUNCTION
    Creates an Alert display with the specified message.

    If the system can recover from this Alert, its 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 THIS: Starting with Version 1.2, if Intuition can’t get enough
    memory to display a RECOVERY_ALERT, the value FALSE will be returned.

    The 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.

    The String argument points to an AlertMessage string.  The AlertMessage
    string is comprised of one or more substrings, each of which is
    comprised 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.

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 is 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.

SEE ALSO
```

## 1.18 intuition.library/DisplayBeep

```
NAME
    DisplayBeep  --  flashes the video display.

SYNOPSIS
    DisplayBeep(Screen)
                A0

    struct Screen *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.

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

RESULT
    None

BUGS

SEE ALSO
```

## 1.19 intuition.library/DoubleClick

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

SYNOPSIS
    IsDouble = DoubleClick(StartSecs, StartMicros, CurrentSecs,
    A0                     D0         D1           D2
                                CurrentMicros)
                                D3
    BOOL IsDouble;
    LONG StartSecs, StartMicros;
    LONG CurrentSecs, CurrentMicros;

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 InputEvents 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()
```

## 1.20 intuition.library/DrawBorder

```
NAME
    DrawBorder  --  draws the specified Border into the RastPort.

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

    struct RastPort *RastPort;
    struct Border     *Border;
    SHORT  LeftOffset, TopOffset;

FUNCTION
    First, sets up the DrawMode 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 to the boundaries of the RastPort's layer, if it exists.
    This is the case with Window RastPorts.

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

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

RESULT
    None

BUGS

SEE ALSO
```

## 1.21 intuition.library/DrawImage

```
NAME
    DrawImage  --  draws the specified Image into the RastPort.

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

    struct RastPort *RastPort;
    struct Image      *Image;
    SHORT  LeftOffset, TopOffset;

FUNCTION
    First, sets up the DrawMode 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 as appropriate -- if
    you draw an image outside of your Window, your imagery will be
    clipped at the Window's edge.

    If the NextImage field of the Image argument is non-zero,
    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

BUGS

SEE ALSO
```

## 1.22 intuition.library/EndRefresh

```
NAME
    EndRefresh  --  Ends the optimized refresh state of the Window.

SYNOPSIS
    EndRefresh(Window, Complete)
               A0      D0

    struct Window *Window;
    BOOL Complete;

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.

    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 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(),
    layers.library/UnlockLayerRom()
```

## 1.23 intuition.library/EndRequest

```
NAME
    EndRequest  --  Ends the Request and resets the Window.

SYNOPSIS
    EndRequest(Requester, Window);
               A0         A1

FUNCTION
    Ends the Request by erasing the Requester and resetting 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()
```

## 1.24 intuition.library/FreeRemember

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

SYNOPSIS
    FreeRemember(RememberKey, ReallyForget)
                 A0           D0

    struct Remember **RememberKey;
    BOOL   ReallyForget;

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.

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()
```

## 1.25 intuition.library/FreeSysRequest

```
NAME
    FreeSysRequest -- Frees resources used by a call to BuildSysRequest().

SYNOPSIS
    FreeSysRequest(Window)
                   A0

    struct Window *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.

    NOTE:  if BuildSysRequest() did not return a pointer to a Window,
    you should not call FreeSysRequest()!

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

RESULT
    None

BUGS

SEE ALSO
    BuildSysRequest(), AutoRequest(), CloseWindow(), exec.library/Wait()
```

## 1.26 intuition.library/GetDefPrefs

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

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

    struct Preferences *Prefs;
    struct Preferences *PrefBuffer;
    SHORT  Size;

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
    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()
```

## 1.27 intuition.library/GetPrefs

```
NAME
    GetPrefs  --  Get the current setting of the Intuition Preferences.

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

    struct Preferences *Prefs;
    struct Preferences *PrefBuffer;

FUNCTION
    Gets a copy of the current Intuition Preferences data.  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.

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()
```

## 1.28 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   Success;
    CPTR   Buffer;
    USHORT Size;
    USHORT Type;
    struct Screen *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.

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

SEE ALSO
    OpenWindow()
```

## 1.29 intuition.library/InitRequester

```
NAME
    InitRequester  --  initializes a Requester structure.

SYNOPSIS
    InitRequester(Requester)
                  A0

    struct Requester *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.

INPUTS
    Requester = a pointer to a Requester structure

RESULT
    None

BUGS

SEE ALSO
```

## 1.30 intuition.library/IntuiTextLength

```
NAME
    IntuiTextLength  --  Returns the length (pixel-width) of an IntuiText.

SYNOPSIS
    IntuiTextLength(IText)
                    D0

    struct IntuiText *IText;

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.

SEE ALSO
    OpenScreen()
```

## 1.31 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        *MenuStrip;
    USHORT MenuNumber;

FUNCTION
    This routine feels through the specified MenuStrip and returns the
    address of the Item specified by the MenuNumber.  Typically,
    you will use this routine to get the address of a MenuItem from
    a MenuNumber sent to you by Intuition after User has played with
    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 MenuStrip, which includes:
       - a valid Menu number
       - a valid Item Number
       - if the Item specified by the above two components has a
         SubItem, the MenuNumber may have a SubItem component too

    Note that there must be BOTH a Menu number and an Item number.
    Because a SubItem specifier is optional, the address returned by
    this routine may point to either an Item or a SubItem.

INPUTS
    MenuStrip = a pointer to the first Menu in your MenuStrip
    MenuNumber = the value which contains the packed data that selects
       the Menu and Item (and SubItem).  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 MenuItem specified
    by MenuNumber.

BUGS

SEE ALSO
```

```
    The "Menus" chapter of the Intuition Reference Manual,
    for more information about "Menu Numbers."
```

## 1.32 intuition.library/LockIBase

```
NAME
    LockIBase -- Intuition user's access to Intuition Locking

SYNOPSIS
    Lock = LockIBase(LockNumber)
    D0               D0

    ULONG Lock;
    ULONG LockNumber;

FUNCTION
    Grabs Intuition internal semaphore so that caller may examine
    IntuitionBase safely.

    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.

INPUTS
    A long unsigned integer, LockNumber, specifies which of Intuition's
    internal locks you want to get.  This parameter should be zero for all
    forseeable 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 should not be called while holding any other system locks
    such as Layer or LayerInfo locks.

SEE ALSO
    UnlockIBase(), layers.library/LockLayerInfo,
    exec.library/ObtainSemaphore
```

## 1.33 intuition.library/MakeScreen

```
NAME
    MakeScreen -- Do an Intuition-integrated MakeVPort() of a custom
```

```
                  screen

SYNOPSIS
    MakeScreen(Screen)
               A0

    struct Screen *Screen;

FUNCTION
    This procedure allows you to do a MakeVPort() for the ViewPort of your
    Custom Screen in an Intuition-integrated way.  This allows you to
    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 is not in use.
        - 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 can call RethinkDisplay() to
    incorporate the new ViewPort of your custom screen into the
    Intuition display.

INPUTS
    Screen = address of the Custom Screen structure

RESULT
    None

BUGS

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

## 1.34 intuition.library/ModifyIDCMP

```
NAME
    ModifyIDCMP  --  Modify the state of the Window's IDCMPFlags.

SYNOPSIS
    ModifyIDCMP(Window, IDCMPFlags)
                A0      D0

    struct Window *Window;
    ULONG  IDCMPFlags;

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 NULL, 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 NULL, 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
   - if the IDCMP for the given Window is opened, and the IDCMPFlags
     argument is not NULL, 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.

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

RESULT
    None

BUGS
    Method for closing a window with a shared port needs to be better
    documented somewhere, or provided as an Intuition call, or both.
    At the present, the technique is available through developer support
    newsletters as a function called CloseWindowSafely().  See, for
    example, Amiga Mail, vol.2.

SEE ALSO
    OpenWindow(), CloseWindow()
```

## 1.35 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

    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Requester;
    USHORT Flags;
    USHORT HorizPot, VertPot;
    USHORT HorizBody, VertBody;

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 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 refinded display updataing, use 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 variable of PropInfo
    HorizPot = value to be stored in the HorizPot variable of PropInfo
    VertPot = value to be stored in the VertPot variable of PropInfo
    HorizBody = value to be stored in the HorizBody variable of PropInfo
    VertBody = value to be stored in the VertBody variable of PropInfo

RESULT
    None

BUGS

SEE ALSO
    NewModifyProp()
    The Intuition Reference Manual contains more information on
    Proportional Gadgets.
```

## 1.36 intuition.library/MoveScreen

```
NAME
    MoveScreen  --  attempts to move the Screen by increments provided.

SYNOPSIS
    MoveScreen(Screen, DeltaX, DeltaY);
               A0      D0      D1

    struct Screen *Screen;
    SHORT  DeltaX, DeltaY;

FUNCTION
    Moves the screen the specified increment.

    Currently, only the DeltaY coordinate is significant; you should
    pass zero for DeltaX.

    Screens are constrained now only by the top and bottom of the
    Intuition View, which is not guaranteed to be the same in all
    versions of the software.

    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 to see where the screen really ended up.

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

INPUTS
    Screen = pointer to a Screen structure
    DeltaX = amount to move the screen on the x-axis
            Note that DeltaX should be set to zero.
    DeltaY = amount to move the screen on the y-axis

RESULT
    None

BUGS

SEE ALSO
    RethinkDisplay()
```

## 1.37 intuition.library/MoveWindow

```
NAME
    MoveWindow  --  Ask Intuition to move a Window.

SYNOPSIS
    MoveWindow(Window, DeltaX, DeltaY)
               A0      D0      D1
    struct Window *Window;
    SHORT  DeltaX, DeltaY;

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.

    This routine does no error-checking.  If your delta values specify
    some far corner of the Universe, Intuition will attempt to move
    your Window to the far corners of the Universe.  Because of the
    distortions in the space-time continuum that can result from this,
    as predicted by special relativity, the result is generally not
    a pretty sight.

    You are thus advised to consider the dimensions of your Window's screen
    and the current position of your window before calling this function.

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

RESULT
    None

BUGS

SEE ALSO
    SizeWindow(), WindowToFront(), WindowToBack()
```

## 1.38 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

    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Requester;
    USHORT Flags;
    USHORT HorizPot, VertPot;
    USHORT HorizBody, VertBody;
    int    NumGad;

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

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 variable of PropInfo
    HorizPot = value to be stored in the HorizPot variable of PropInfo
    VertPot = value to be stored in the VertPot variable of PropInfo
    HorizBody = value to be stored in the HorizBody variable of PropInfo
    VertBody = value to be stored in the VertBody variable of 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.
```

## 1.39 intuition.library/OffGadget

```
NAME
    OffGadget  --  disables the specified Gadget.

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

    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Requester;

FUNCTION
    This command disables the specified Gadget.  When a Gadget is
    disabled, these things happen:
         - its imagery is displayed ghosted
         - the GADGDISABLED 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
    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.

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 by NULL if this isn't
       a Requester Gadget (i.e. REQGADGET is not set)).

RESULT
    None

BUGS

SEE ALSO
    AddGadget(), RefreshGadgets()
```

## 1.40 intuition.library/OffMenu

```
NAME
    OffMenu  --  disables the given menu or menu item.

SYNOPSIS
    OffMenu(Window, MenuNumber)
            A0      D0

    struct Window *Window;
    USHORT MenuNumber;

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
```

## 1.41 intuition.library/OnGadget

```
NAME
    OnGadget  --  enables the specified Gadget.

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

    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Requester;

FUNCTION
    This command enables the specified Gadget.  When a Gadget is
    enabled, these things happen:
    - its imagery is displayed normally (not ghosted)
    - the GADGDISABLED 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
    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.

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 by NULL if this isn't
       a Requester Gadget (i.e. REQGADGET is not set)).

RESULT
    None

BUGS

SEE ALSO
```

## 1.42 intuition.library/OnMenu

```
NAME
    OnMenu  --  disables the given menu or menu item.

SYNOPSIS
    OnMenu(Window, MenuNumber)
```

```
          A0       D0

    struct Window *Window;
    USHORT MenuNumber;

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
```

## 1.43 intuition.library/OpenScreen

```
NAME
    OpenScreen  --  Open an Intuition Screen.

SYNOPSIS
    Screen = OpenScreen(NewScreen)
    D0                  A0

    struct Screen *Screen;
    struct NewScreen *NewScreen;

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.
    That structure is initialized with the following information:
    -------------------------------------------------------------------------
    Left = initial x-position of your Screen (should be zero currently)
    Top = initial y-position of the opening Screen
    Width = the width for this Screen’s RastPort.
    Height = the height for his Screen’s RastPort, or the constant
             STDSCREENHEIGHT to get current local maximum (at this time
             guaranteed to be at least 200).  The actual height the screen
             opended to can be found in the returned Screen structure.
    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.

    Depth = number of BitPlanes
    DetailPen = pen number for details (like gadgets or text in title bar)
    BlockPen = pen number for block fills (like title bar)
    Type = Screen type
        Set these flag bits 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 it’s 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 RMBTRAP flag on all your
            screen’s windows, this flag will prevent Intuition from
            rendering into your screen’s bitplanes.  Without RMBTRAP (or
            using MENUVERIFY IDCMP facility to cancel menu operations),
            this flag will prevent Intuition from clearing your menu bar,
            which is probably unacceptable.  The title bar layer may still
            overwrite your bitmap on open.
    ViewModes = the appropriate argument for the data type ViewPort.Modes.
           these might include:
                  HIRES for this screen to be HIRES width.
                  INTERLACE for the display to switch to interlace.
                  SPRITES for this Screen to use sprites (pointer comes
                      anyway).
                  DUALPF for dual-playfield mode (not supported yet)
    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.
    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
    Gadgets = This field should be set to NULL, since no user Gadgets may
       be attached to a Screen.
    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 argument should
      point to the BitMap that describes your display memory.  See the
      "Screens" chapter and the "Amiga ROM Kernel Manual" for more
      information about BitMaps.

RESULT
```

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

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 availble
    in the AmigaDOS books.  Sample code fragment:

    ----------- cut here ----------
    #include "libraries/dosextens.h"
            ...
    struct Process *process;
    struct Window  *window;
    APTR           temp;
            ...
        process=(struct Process *)FindTask(0L);
        temp=process->pr_WindowPtr;         /* save old value */
        process->pr_WindowPtr=(APTR)window;
        /* set a pointer to any open window on your screen */
            ...
            your code goes here
            ...
        process->pr_WindowPtr=temp;
        /* restore value _before_ CloseWindow */
        CloseWindow(window);
    ------- cut here ------

BUGS

SEE ALSO
    OpenWindow(), PrintIText(), CloseScreen(), The Intuition Reference
    Manual
```

## 1.44 intuition.library/OpenWindow

```
NAME

OpenWindow  --  Opens an Intuition Window

SYNOPSIS

OpenWindow(NewWindow);
where the NewWindow structure is initialized with:
    Left, Top, Width, Height, DetailPen, BlockPen, Flags,
    IDCMPFlags, Gadgets, CheckMark, Text, Type, Screen, BitMap,
    MinWidth, MinHeight, MaxWidth, MaxHeight

FUNCTION

Opens an Intuition window of the given height, width and depth, including
the specified system Gadgets as well as any of your own.  Allocates
everything you need to get going.
```

```
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.

If the flag SUPER_BITMAP 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.

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)

Flags = specifiers for your requirements of this window, including:
        - which system Gadgets you want attached to your window:
            - WINDOWDRAG allows this Window to be dragged
            - WINDOWDEPTH lets the user depth-arrange this Window
            - WINDOWCLOSE attaches the standard Close Gadget
            - WINDOWSIZING allows this Window to be sized.  If you ask
              the WINDOWSIZING Gadget, you must specify one or both
              of the flags SIZEBRIGHT and SIZEBBOTTOM below; if you
              don't, the default is SIZEBRIGHT.  See the
              following items SIZEBRIGHT and SIZEBBOTTOM for extra
              information.
            - SIZEBRIGHT 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 SIZEBBOTTOM 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
              for instance, applications that want every possible
              horizontal bit and are willing to lose lines vertically.
              NOTE:  if you select WINDOWSIZING, you must select
              either SIZEBRIGHT or SIZEBBOTTOM or both.  If you select
              neither, the default is SIZEBRIGHT.
            - SIZEBBOTTOM 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
              SIZEBRIGHT above.
              NOTE:  if you select WINDOWSIZING, you must select
              either SIZEBRIGHT or SIZEBBOTTOM or both.  If you select
              neither, the default is SIZEBRIGHT.
        - GIMMEZEROZERO for easy but expensive output
        - what type of window raster you want, either:
            - SIMPLE_REFRESH
            - SMART_REFRESH
            - SUPER_BITMAP
          If the type is SMART_REFRESH, and you do not handle
          REFRESHWINDOW type messages, also set the NOCAREREFRESH
          flag.
        - BACKDROP for whether or not you want this window to be one
          of Intuition's special backdrop windows.  See BORDERLESS
          as well.
        - 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 (GIMMEZEROZERO notwithstanding).
          This flag can work in conjunction with the IDCMP Flag
          called MOUSEMOVE, which allows you to listen via the
          IDCMP.
        - 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 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.
        - 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
          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.
        - 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.

IDCMPFlags = IDCMP is the acronym for Intuition Direct Communications
     Message Port.  It's Intuition's sole acronym, given in honor of
     all hack-heads who love to mangle our brains with maniacal names,
     and fashioned especially cryptic and unpronounceable to make them
     squirm with sardonic delight.  Here's to you, my chums.  Meanwhile,
     I still opt (and argue) for simplicity and elegance.
         If any of the IDCMP Flags is selected, Intuition will create
     a pair of messageports 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 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 me that if a given event occurs which your
     program wants to know about, I'm to broadcast the details
     of that event through the IDCMP rather than via the Console device.
     device.  This allows a program to interface with Intuition
     directly, rather than going through the Console device.
         Remember, if you are going to open both an IDCMP and
     a Console, it will be far better to get most of the event
     messages via the Console.  Reserve your usage of the IDCMP
     for special performance cases; that is, when you aren't going
     to open a Console for your Window and you do want to learn
     about a certain set of events (for instance, CLOSEWINDOW); another
     example would be SIZEVERIFY, which is a function that you get
     ONLY through the use of the IDCMP (because the Console doesn't
     give you any way to talk to Intuition directly).
         On the other hand, if the IDCMPFlags argument is equal to
     zero, no IDCMP is created and the only way you can learn about any
     Window event for this Window is via a Console opened for
     this Window.  And you have no way to SIZEVERIFY.
```

```
    If you want to change the state of the IDCMP some time after
you’ve opened the Window (including opening or closing the IDCMP)
you call the routine ModifyIDCMP().
    The flags you can set are:
     - REQVERIFY is the flag which, like SIZEVERIFY and(see
       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 into that Window, 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.
     - REQCLEAR is the flag you set to hear about it when the
       last Requester is cleared from your Window and
       it’s safe for you to start output again (presuming you’re
       using REQVERIFY)
     - REQSET is a flag that you set to receive a broadcast
       when the first Requester is opened in your Window.
       Compare this with REQCLEAR above.  This function is
       distinct from REQVERIFY.  This functions merely tells you
       that a Requester has opened, whereas REQVERIFY requires
       you to respond before the Requester is opened.
     - 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.
     - 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 SIZEVERIFY message and
       Wait() until you reply that it’s OK to proceed with
       the sizing. NOTE:  when I say that Intuition will
       Wait() until you reply, what I’m 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 NEWSIZE:


       With all of the "VERIFY" functions, it is not safe
       to leve them enabled at any time when you 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 AmigaDOS!!!


    - 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
    - REFRESHWINDOW when set will cause a Message to be sent
      whenever your Window needs refreshing.  This flag makes
      sense only with SIMPLE_REFRESH and SMART_REFRESH Windows.
    - 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).
    - MOUSEMOVE will work only if you've set the flag
      REPORTMOUSE above, or if one of your Gadgets has the
      flag FOLLOWMOUSE set.  Then all mouse movements will be
      reported here.
    - GADGETDOWN means that when the User "selects" a Gadget
      you've created with the GADGIMMEDIATE flag set, the fact
      will be broadcast through the IDCMP.
    - GADGETUP means that when the User "releases" a Gadget that
      you've created with the RELVERIFY flag set, the fact
      will be broadcast through the IDCMP.
    - MENUPICK selects that MenuNumber data will come this way
    - CLOSEWINDOW means broadcast the CLOSEWINDOW event through
      the IDCMP rather than the Console
    - RAWKEY selects that all RAWKEY events are transmitted via
      the IDCMP.  Note that these are absolutely RAW keycodes,
      which you will have to massage 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.

    - VANILLAKEY is for developers who don't want the hassle
      of RAWKEYS.  This flag will return all the keycodes
      after translation via the current country-dependant keymap.
      When you set this flag, you will get IntuiMessages where the
      Code field has a decoded ASCII character representing the key
      struck on the keyboard.  Only codes that map to one character
      are returned, you can't read such keys as HELP or the Function
      keys with VANILLAKEY.

    - 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 INTUITICKS message
            and haven't replied to it, another message will not be sent.
            Intuition receives timer events ten times a second
            (approximately).

          - DELTAMOVE gives raw (unscaled) input event delta X/Y values.
            This is so you can detect mouse motion regardless of
            screen/window/display boundaries. Note that MOUSEBUTTONS
            messages will also be affected.

          - NEWPREFS indicates you wish to be notified when the system-
            wide preferences changes.

          - Set ACTIVEWINDOW and INACTIVEWINDOW to get messages when those
            events happen to your window.  Take care not to confuse this
            "ACTIVEWINDOW" with the remarkably familiar sounding, but
            totally different "WINDOWACTIVE" flag.

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
      MenuItems 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

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 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
       SUPER_BITMAP, this pointer is ignored

MinWidth, MinHeight, MaxWidth, MaxHeight = the size limits for this
      that the minimums cannot be greater than the current size,
      nor can the maximums be smaller than the current size.

      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 the maximum available.

      Any one of these can be initialized to zero, which means 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.

RESULT

If all is well, returns the pointer to your new Window

If anything goes wrong, returns NULL

BUGS

SEE ALSO

OpenScreen(), ModifyIDCMP(), WindowTitles()
```

## 1.45 intuition.library/OpenWorkBench

```
NAME
    OpenWorkBench  --  Opens the WorkBench Screen

SYNOPSIS
    WBScreen =  OpenWorkBench()
    D0

    struct Screen *WBScreen;

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-FALSE if WorkBench Screen opened successfully, or was already
    opened FALSE if anything went wrong and the WorkBench Screen isn't out
    there

BUGS
```

```
SEE ALSO
```

## 1.46 intuition.library/PrintIText

```
NAME
    PrintIText  --  prints the text according to the IntuiText argument

SYNOPSIS
    PrintIText(RastPort, IText, LeftOffset, TopOffset)
               A0        A1     D0          D1

    struct RastPort *RastPort;
    struct IntuiText *IText;
    SHORT LeftOffset, TopOffset;

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 IntuitText 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.

    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 ITextAttr 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
```

## 1.47 intuition.library/RefreshGadgets

```
NAME
    RefreshGadgets  --  Refresh (redraw) the Gadget display

SYNOPSIS
    RefreshGadgets(Gadgets, Window, Requester)
                   A0       A1      A2

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
    REQGADGET flag set, a requirement for all Requester Gadgets.

    The Pointer argument points 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 SELECTED flag of a
    Boolean Gadget and then call RefreshGadgets() to see them displayed
    highlighted if and only if SELECTED is set.  If you wish to do this
    and be completely proper, you must RemoveGadget(), change SELECTED
    flag, AddGadget(), and RefreshGadgets(), or the equivalent.

    The Gadgets argument can be a copy of the FirstGadget variable in
    either the Screen or 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-GIMMEZEROZERO 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()
```

## 1.48 intuition.library/RefreshGList

```
NAME
    RefreshGList  --  Refresh (redraw) a chosen number of gadgets.

SYNOPSIS
    RefreshGList(Gadgets, Window, Requester, NumGad)
                 A0       A1      A2         D0

    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Requester;
    SHORT  NumGad;

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 variable can point to a Requester structure.  If
    the first Gadget in the list has the 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 simple 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 Gadget
      is a Requester Gadget (REQGADGET) ALL gadgets in the requester
      will be refreshed (this is a mode compatible with v1.1
      RefreshGadgets().
```

```
RESULT
    None

BUGS

SEE ALSO
    RefreshGadgets()
```

## 1.49 intuition.library/RefreshWindowFrame

```
NAME
    RefreshWindowFrame -- Ask Intuition to redraw your window
                          border/gadgets

SYNOPSIS
    RefreshWindowFrame(Window)
                       A0

    struct Window *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
```

## 1.50 intuition.library/RemakeDisplay

```
NAME
    RemakeDisplay  --  Remake the entire Intuition display

SYNOPSIS
    RemakeDisplay()

FUNCTION
    This is the big one.

    This procedure remakes the entire Intuition display.  It does
    the equivalent of MakeScreen() for every Screen in the system,
    and then it calls RethinkDisplay().

    WARNING:  This routine can take several milliseconds to run, so
    do not use it lightly.  RethinkDisplay() (called by this routine)
    does a Forbid() on entry and a Permit() on exit, which can seriously
    degrade the performance of the multi-tasking Eexecutive.

INPUTS
    None

RESULT
    None

BUGS

SEE ALSO
    MakeScreen(), RethinkDisplay(), graphics.library/MakeVPort
```

## 1.51 intuition.library/RemoveGadget

```
NAME
    RemoveGadget  --  removes a Gadget from a Window

SYNOPSIS
    Position = RemoveGadget(Window, Gadget)
    D0                      A0      A1

    USHORT Position;
    struct Window *Window;
    struct Gadget *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.

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(), RemoveGList()
```

## 1.52 intuition.library/RemoveGList

```
NAME
    RemoveGList  --  removes a sublist of Gadgets from a Window.

SYNOPSIS
    Position =  RemoveGList(Window, Gadget, Numgad)
    D0                      A0      A1      D0

    struct Window *Window;
    struct Gadget *Gadget;
    SHORT  Numgad;

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 REQGADGET is set.

    Otherwise identical to RemoveGadget().

NOTE
    The last gadget in the list does NOT have it's link zeroed.

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()
```

## 1.53 intuition.library/ReportMouse

```
NAME
    ReportMouse  --  tells Intuition whether to report mouse movement.
```

```
SYNOPSIS
    ReportMouse(Boolean, Window)
                D0       A0         <-note
    BOOL   Boolean;
    struct Window *Window;

SPECIAL NOTE
    Some compilers and link files switch the arguments to this function
    about in unpredictable ways.  The call will take one of two forms:

            ReportMouse(Window, (ULONG)Boolean);
                    -or-
            ReportMouse(Boolean, Window);

    The Manx Aztec compiler prefers the second form.  From assembler the
    interface is always the same:  Boolean in D0, Window in A0

    Also, it is still endorsed to simply set the REPORTMOUSE flag bit
    in Window->Flags, or reset it, on your own.  Make the operation
    an atomic assembly instruction (e.g.: OR.W #REPORTMOUSE,wd_Flags+2(A0)
    where A0 contains your window pointer).  Most compilers will produce
    an atomic operation when faced with:
                    Window->Flags |= REPORTMOUSE;
                    Window->Flags &=~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 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
    FOLLOWMOUSE flag is examined anew.

    Note also that calling ReportMouse() when no Gadget is currently
    selected will change the state of the Window's REPORTMOUSE flag, but
    will have no effect on any Gadget that may be subsequently selected.

    The ReportMouse() function is first performed when OpenWindow()
    is first called; if the flag 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 REPORTMOUSE is not set, then no mouse-movement reports will
    be broadcast until ReportMouse() is called with a Boolean of TRUE.

    Note that the 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 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
```

## 1.54 intuition.library/Request

```
NAME
    Request  --  Activates a Requester.

SYNOPSIS
    Success = Request(Requester, Window);
    D0                A0         A1

    BOOL Success;
    struct Requester *Requester;
    struct Window *Window;

FUNCTION
    Links in and displays a Requester into the specified Window.

    This routine ignores the Window's REQVERIFY flag.

INPUTS
    Requester = pointer to the Requester to be displayed
    Window = pointer to the Window into which this Requester goes

RESULT
    If the Requester is successfully opened, TRUE is returned.  Otherwise,
    if the Requester could not be opened, FALSE is returned.

BUGS
    POINTREL requesters not currently supported, by THIS call, but
    are now supported for Double-Menu Requesters.

SEE ALSO
    The Requesters section of the Intuition Reference Manual
```

## 1.55 intuition.library/RethinkDisplay

```
NAME
    RethinkDisplay  --  the grand manipulator of the entire Intuition
                        display
```

```
SYNOPSIS
    RethinkDisplay()

FUNCTION
    This function performs the Intuition global display reconstruction.
    This includes rethinking about all of the  ViewPorts and their
    relationship to another and reconstructing the entire display based
    on the results of this rethinking.

    Specifically, and omitting some internal details, the operation
    consists of this:

        Determine which ViewPorts are invisible and set their VP_HIDE
        ViewPort Mode flag.

        If a change to a viewport height or changing interlace needs
        require, MakeVPort() is called for specific ViewPorts.  After
        this phase, the 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 best left to Intuition.

    WARNING:  This routine can take several milliseconds to run, so
    do not use it lightly.  RethinkDisplay() does a Forbid() on entry
    and a Permit() on exit, which can seriously degrade the performance
    of the multi-tasking Eexecutive.

INPUTS
    None

RESULT
    None

BUGS

SEE ALSO
    RemakeDisplay(), graphics.library/MakeVPort(),
    graphics.library/MrgCop(), graphics.library/LoadView(),
    MakeScreen()
```

## 1.56 intuition.library/ScreenToBack

```
NAME
    ScreenToBack  --  send the specified Screen to the back of the display.

SYNOPSIS
    ScreenToBack(Screen)
                 A0
    struct Screen *Screen;

FUNCTION
    Sends the specified Screen to the back of the display.

INPUTS
    Screen = pointer to a Screen structure

RESULT
    None

BUGS

SEE ALSO
```

## 1.57 intuition.library/ScreenToFront

```
NAME
    ScreenToFront  --  brings the specified Screen to the front of the
                       display

SYNOPSIS
    ScreenToFront(Screen)
                  A0

FUNCTION
    Brings the specified Screen to the front of the display.

INPUTS
    Screen = a pointer to a Screen structure

RESULT
    None

BUGS

SEE ALSO
```

## 1.58 intuition.library/SetDMRequest

```
NAME
    SetDMRequest  --  sets the DMRequest of the Window.

SYNOPSIS
    SetDMRequest(Window, DMRequester)
                 A0      A1

    struct Window *Window;
    struct Requester *DMRequester;

FUNCTION
```

```
    Attempts to set the DMRequester into the specified window.
    The DMRequester 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 set the DMRequester
    if it's already set and is currently active (in use by the user).
    After having called SetDMRequest(), if you want to change the
    DMRequester, the correct way to start is by calling ClearDMRequest()
    until it returns a value of TRUE; then you can call SetDMRequest()
    with the new DMRequester.

    If the POINTREL flag is set, 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
    DMRequester = 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()
```

## 1.59 intuition.library/SetMenuStrip

```
NAME
    SetMenuStrip  --  Attaches the Menu strip to the Window.

SYNOPSIS
    Success = SetMenuStrip(Window, Menu)
    D0                     A0      A1

    BOOL   Success;
    struct Window *Window;
    struct Menu *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 MenuItems 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()
```

## 1.60 intuition.library/SetPointer

```
NAME
    SetPointer  --  sets a Window with its own Pointer

SYNOPSIS
    SetPointer(Window, Pointer, Height, Width, XOffset, YOffset)
               A0      A1       D0      D1     D2       D3

    struct Window *Window;
    USHORT  *Pointer;
    SHORT Height, Width;
    SHORT XOffset, YOffset;

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 its version of the Pointer.  If the
    Window is the active one when this routine is called, the
    change takes place immediately.

    The XOffset and YOffset are used to offset the top-left corner
    of the hardware sprite imagery 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 Pointer 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 Pointer position.
    If you specify an XOffset of -15, the right-edge of the sprite
    will be over the Pointer 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 Pointer position
    YOffset = the offset for your sprite from the Pointer position

RESULT
    None

BUGS

SEE ALSO
    ClearPointer()
```

## 1.61 intuition.library/SetPrefs

```
NAME
    SetPrefs  --  Set Intuition Preferences.

SYNOPSIS
    Prefs = SetPrefs(PrefBuffer, Size, Inform)
    D0               A0          D0    D1

    struct Preferences *Prefs;
    struct Preferences *PrefBuffer;
    int    Size;
    BOOL   Inform;

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 a NEWPREFS
    message is to be sent to all Windows that have the 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.

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 propogated to all windows.

RESULT
    Returns your parameter PrefBuffer.

BUGS
```

```
SEE ALSO
    GetDefPrefs(), GetPrefs()
```

## 1.62 intuition.library/SetWindowTitles

```
NAME
    SetWindowTitles  --  Sets the Window's titles for both Window and
                         Screen

SYNOPSIS
    SetWindowTitles(Window, WindowTitle, ScreenTitle)
                    A0      A1           A2

    struct Window *Window;
    UBYTE *WindowTitle, *ScreenTitle;

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. (struct Window *) ~0) for either
    of the title pointers.  This designates that you want to 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()
```

## 1.63 intuition.library/ShowTitle

```
NAME
    ShowTitle  --  Set the Screen title bar display mode

SYNOPSIS
    ShowTitle(Screen, ShowIt)
              A0      D0

    struct Screen *Screen;
    BOOL   ShowIt;

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 BACKDROP
    Windows.  This is contrasted with the fact that non-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 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 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
```

## 1.64 intuition.library/SizeWindow

```
NAME
    SizeWindow  --  Ask Intuition to size a Window.

SYNOPSIS
    SizeWindow(Window, DeltaX, DeltaY)
               A0      D0      D1

    struct Window *Window;
    SHORT  DeltaX, DeltaY;

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
    you Window has finally been sized by setting the 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.

    This routine does no error-checking.  If your delta values specify
    some far corner of the Universe, Intuition will attempt to size
    your Window to the far corners of the Universe.  Because of the
    distortions in the space-time continuum that can result from this,
    as predicted by special relativity, the result is generally not
    a pretty sight.

INPUTS
    Window = pointer to the structure of the Window to be sized
    DeltaX = signed value describing how much to size Window on the x-axis
    DeltaY = signed value describing how much to size Window on the y-axis

RESULT
    None

BUGS

SEE ALSO
    MoveWindow(), WindowToFront(), WindowToBack()
```

## 1.65 intuition.library/UnlockIBase

```
NAME
    UnlockIBase -- surrender an Intuition lock gotten by LockIBase()

SYNOPSIS
    UnlockIBase(Lock)
                A0

    ULONG Lock;
```

```
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()
```

## 1.66 intuition.library/ViewAddress

```
NAME
    ViewAddress  --  Returns the address of the Intuition View structure.

SYNOPSIS
    ViewAddress()

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
```

## 1.67 intuition.library/ViewPortAddress

```
NAME
    ViewPortAddress  --  Returns the address of a Window’s ViewPort
                         structure.
```

```
SYNOPSIS
    ViewPortAddress(Window)
                    A0

    struct Window *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.

INPUTS
    Window = pointer to the Window for which you want the ViewPort address

RESULT
    Returns the address of the Intuition View structure

BUGS

SEE ALSO
    graphics.library
```

## 1.68 intuition.library/WBenchToBack

```
NAME
    WBenchToBack  --  Sends the WorkBench Screen in back of all Screens.

SYNOPSIS
    Success = WBenchToBack()
    D0

    BOOL Success;

FUNCTION
    Causes the WorkBench Screen, if it's currently opened, to go to
    the background.  This does not 'move' the Screen up or down, instead
    only affects the depth-arrangement of the Screen.

    If the WorkBench Screen was opened, this function returns TRUE,
    otherwise it returns FALSE.
```

```
INPUTS
    None

RESULT
    If the WorkBench Screen was opened, this function returns TRUE,
    otherwise it returns FALSE.

BUGS

SEE ALSO
    WBenchToFront(), ScreenToFront()
```

## 1.69 intuition.library/WBenchToFront

```
NAME
    WBenchToFront  --  Brings the WorkBench Screen in front of all Screens.

SYNOPSIS
    Success = WBenchToFront()
    D0

    BOOL Success;

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.

    If the WorkBench Screen was opened, this function returns TRUE,
    otherwise it returns FALSE.

INPUTS
    None

RESULT
    If the WorkBench Screen was opened, this function returns TRUE,
    otherwise it returns FALSE.

BUGS

SEE ALSO
    WBenchToBack(), ScreenToBack()
```

## 1.70 intuition.library/WindowLimits

```
NAME
    WindowLimits  --  Set the minimum and maximum limits of the Window.

SYNOPSIS
    Success = WindowLimits(Window, MinWidth, MinHeight, MaxWidth,
    D0                     A0      D0        D1         D2
         MaxHeight)
         D3
    BOOL   Success;
    struct Window *Window;
    SHORT  MinWidth, MinHeight;
    USHORT MaxWidth, MaxHeight;

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 Window’s
    initial size, which means that the user won’t be able to size it at
    all.  After the 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.

    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()
```

## 1.71 intuition.library/WindowToBack

```
NAME
    WindowToBack  --  Ask Intuition to send this Window to the back

SYNOPSIS
    WindowToBack(Window)
                 A0

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 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()
```

## 1.72 intuition.library/WindowToFront

```
NAME
    WindowToFront  --  Ask Intuition to bring this Window to the front.

SYNOPSIS
    WindowToFront(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 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()
```