Apig

COLLABORATORS

	TITLE : Apig		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Apig

1.1 Apig.guide

```
This is the documents for Apig.library in AmigaGuide format.


Author~says:

Preface

Using~the~library
          Important!


Functions

Sorted~functions

Function~reference

Sorted~function~reference

Error~codes

Menu~Hints

Something~about~APIG~pointers

Some~words~from~the~creator~of~this~guide
```

1.2 Author says

```
This program is in the public domain, it may be freely distributed
and copied without restriction.

          Ronnie E. Kelly
```

1.3 Preface to the library

This library provides programmers with a means to access most of the Intuition/Graphic library functions from ARexx. The library functions allow you to specify most of the parameters used in the underlying data structures of the Intuition/Graphic routines. Using the parameters you specify the APIG library builds an instance of the data structure and passes a pointer to it back to you. Hopefully you will be able to create more custom looking interface for your ARexx applications.

The APIG library, for the most part, simply converts the ARexx string parameters to the appropriate form the Intuition/Graphic functions expects and then makes a call directly to the corresponding Intuition/ Graphic library function. Few assumptions have been made about how you will use these functions, therefore this library does very little validating of the inputs you supply.

The library supports the following structures:

```
     Menus, MenuItems, SubItems
     Requesters
     Boolean, String, and Proportional gadgets
     Borders
     IntuiText
     Arrays of short (16-bit) integers
     Layers
     Images  ( via iff.library )
```

APIG uses Christian Weber's iff.library for IFF support, you will need the iff.library (version 18) if you plan to use any of the IFF functions.

If you are not familiar with the Intuition/Graphic library functions or the data structures used by these functions, then I would strongly recommend obtaining a copy of the latest revision of the RKM manual (Autodocs & Includes) and/or the Intuition Reference manual.

1.4 Using the library

Using The Library:

Like all libraries the APIG library must first be placed in your 'LIBS:' directory, the iff.library should be placed in 'LIBS:' as well.

Before calling any of the APIG library functions you must add the APIG library to ARexx's external library function list. To do this include the following statement at the beginning of your ARexx program:

```
         call addlib("apig.library",0,-30,0)
```

You do not need to do this for the iff.library, APIG will open/close

```
 the iff.library when needed.  APIG does not have a 'color requester' or
 a 'filerequester' function and I have no plans of adding them.  However
 you can still use WGL's rexxarplib.library or whatever, if you want.
 In the case of rexxarplib.library, their are a couple of function
 name conflicts (OpenScreen and OpenWindow).  You can resolve these as
 prescribed in the ARexx manual by using the ARexx RemLib() function.
 ie. If you are gonna start calling functions in rexxarplib.library
 you may want to do:

      call remlib('apig.library')

      removing apig.library from the external function list, then be sure
      to add it back on when you go to call an APIG library function.
      (you really only need to do the remlib() if the function you are
       calling conflicts with some other library function name)



 You are now ready to use to the APIG library functions.
  ------

 Most of the functions have the same name as their Amiga Intuition/Graphics
 library counterpart and are used the same way.  The number of parameters,
 their order, and their type are the same as described in the RKM manual.

 eg.
     The Intuition function DrawImage() expects a pointer to a RastPort,
     a pointer to an Image structure, and X & Y coordinates.
     The APIG library function DrawImage() expects the same parameters,
     the same type, and in the same order as the Intuition function.
     Unless otherwise noted in the function descriptions below this will
     always be the rule.

All function parameters must be specified and must be of the type expected
by the function.  If you specify more parameters than the function expects
the superflous parameters will be ignored.  If you specify fewer parameters
than the function expects, then the function will return with a ARexx
error code of 17, your ARexx program will terminate as well.

BIG WARNING !!!  Parameters specified in the wrong order or wrong type
will more than likely bring on the GURU.

If your ARexx program terminates with windows/screens still open it may
still be possible to close the windows/screens and recover the memory
used by them, see the OPENWINDOW() function description on how this is
done.

The APIG functions always return a result to you, even if the Intuition,
Graphic, or Layer function returns no result.  If the Amiga library
function returns a result, then the result returned is that result.

If the Amiga library function does not return a result, then the value
returned to you is either 1 or 0.  A return value of 1 indicates
that the APIG library made the call to the Amiga library function.
A return value of 0 indicates that the Amiga library function was not
called (possibly due to error).
```

Their are several functions in the library for building instances of the various Amiga graphic structures. Use these functions to create the pointers (ARexx 'hex' strings) needed by the other functions. Functions which return pointers, either return a valid (valued) hex string or they return NULL (ie. '0000 0000'x). You should verify that you do indeed have a valid (non-null) pointer before using it. The library checks for null pointers and will not perform a function if it expects the pointer argument(s) to be non-null. Valid pointers are the actual address of the data structure created. All fields within the structure are available for your use (via GETVALUE/SETVALUE), field offsets are as defined in the RKM's for 1.3. Their is only ONE exception, the field ExtData in the Window structure has been appropriated by the APIG library. Do not modify in any way the ExtData field in the Window structure.

Many of the graphic data structures contain 'flags' which describe certain characteristics of the structure. The APIG library uses the same flag values as defined in the RKM manuals, without exception. (see SET_APIG_GLOBALS function description below)

Where multiple flag values are required (needed) you can simply sum all the values together. The flag values you specify are not 'touched' and get passed directly to the Amiga library function/data structure. Their are very few defaults so be sure to specify all that you need.

The APIG library should return all memory allocated to the system memory pool. Version 1.1 of APIG allocates memory for the structures differently than in version 0.5.

IN VERSION 0.5 OF APIG, GADGETS, INTUITEXT, REQUESTERS, MENUS AND SUCH WERE ALLOCATED TO A SPECIFIC WINDOW AND EXISTED UNTIL THE WINDOW WAS CLOSED. THEIR WERE NO FUNCTIONS IN VERSION 0.5 FOR FREEING THE MEMORY USED BY THESE STRUCTURES, SINCE ALL MEMORY WAS 'OWNED' BY THE WINDOW.

IN VERSION 1.1 OF APIG, GADGETS, INTUITEXT, REQUESTERS, MENUS AND SUCH CAN NOW BE INDEPENDENTLY FREED PRIOR TO CLOSING THE WINDOW. MENUS NO LONGER NEED BE RE-CREATED EACH TIME A WINDOW IS CLOSED AND THEN RE-OPENED.

IN FACT MOST OF THE DATA STRUCTURES CAN BE ALLOCATED INDEPENDENTLY AND, CAN BE 'OWNERS' OF OTHER STRUCTURES. STRUCTURES WHICH ARE ALLOCATED TO (OWNED BY) A WINDOW ARE FREED WHEN THE WINDOW CLOSES, SIMILARLY STRUCTURES WHICH ARE 'OWNED' BY OTHER STRUCTURES ARE FREED WHEN THEIR 'OWNER' IS FREED.

THE USE OF AN 'OWNER' SIMPLY ALLOWS YOU TO BIND THE EXISTENCE OF THE STRUCTURE TO THE LIFE OF ITS OWNER. WHEN THE OWNER IS FREED, EVERYTHING IT OWNS IS ALSO FREED. THUS IF YOU WANTED, YOU COULD MAKE A WINDOW THE OWNER OF EVERYTHING YOU ALLOCATE, WHEN THE WINDOW IS CLOSED ALL MEMORY ALLOCATED, GADGETS, INTUITEXT, REQUESTERS, BORDERS, ETC. WILL BE FREED AS WELL. THIS ALSO ALLOWS YOU TO HAVE MULTIPLE WINDOWS OWNING WHAT THEY NEED AND SHARING INDEPENDENT STRUCTURES, WHICH REMAIN VALID AFTER THE WINDOW IS CLOSED.

You should find that this relieves you of having to keep track of all the structures you have allocated then having to explicity free each one. By logical (planned) allocation of structures to an 'owner' you can reduce the amount of memory being used by structures that are no longer needed.

```
     eg. creating a requester

     req  = makerequester(0, ... )         /* 0 = independent       */

     gad1 = makeboolgadget( req,...,req )  /* req owns this gadget  */
                                           /* and link it to req    */

     gad2 = makestrgadget( req,...,req )   /* req owns this gadget  */
                                           /* and link it to req    */

     gad3 = makepropgadget( req,...,req )  /* req owns this gadget  */
                                           /* and link it to req    */

     gad4 = makeboolgadget( 0,...,gad3 )   /* 0 = independent       */

     txt1 = makeitext( req,...,req  )      /* req owns this itext   */
                                           /* and link it to req    */

     txt2 = makeitext( req,...,txt1 )      /* req owns this itext   */
                                           /* and link it to txt1   */
                                           /* which is the same as  */
                                           /* linking it to req     */

     txt3 = makeitext( req,...,txt2 )      /* req owns this itext   */
                                           /* and link it to txt2   */

     (linking here means the gadget/itext is added to the
      gadget/itext list in the structure, see function
      descriptions below)
```

The requester created is independent of any other structure, and must be explictly freed. It need not be re-created each time and can be put in any window (via REQUEST()) you happen to open. When done with the window, the window can close, but the requester will still be useable.

Now when the requester is freed the gadgets, gad1, gad2, and gad3 will also be freed, the intuitext, txt1, txt2, txt3 will be freed as well.

Gadget gad4 will not be freed, it is independent, even though it has been linked to gadget gad3. This is unsafe,so dont do it.

With one single ’free’ call all the structures needed by the requester are freed with it. The gadgets and intuitext owned by the requester cannot be independently freed. APIG will not be able to find the gadgets and intuitext owned by the requester on the lists APIG maintains of allocated structures. APIG could find them but it would have to search each and every memory BLOCK that it allocated. In the above situation APIG would only see two STRUCTURE allocations (req and gad4) but it wont search the memory BLOCKS owned by them.

```
 Now their are some restrictions on what can be an owner and what
 can be owned, they are simply

     1).   A WINDOW is ALWAYS independent and can own ANYTHING,
           except a screen.  (a window can never be 'owned')

     2).   A SCREEN is ALWAYS independent and can own ANYTHING,
           except a window.  (a screen can never be 'owned')

     3).   GADGETS, INTUITEXT, REQUESTERS, and BORDERS can all be
           owners and all can be owned. However they cannot
           own a menu/menuitem/subitem. A border can own a
           requester, though it makes more sense for a requester
           to own a border.

     4).   MENU structures can own ANYTHING that can be owned,
           including, gadgets, intuitext, requesters, borders, etc.
           Menus can only be owned by another menu, a window, or
           a screen.

     5).   MENUITEMS and SUBITEMS cannot be owners, and can only
           be owned by a menu, a window, or screen.

     6).   BITMAPS, RASTPORTS, and IMAGES, are the same as (3) above.
           An image can own Intuitext, Gadget, etc. but cannot own
           a menu/menuitem/subitem.

Basically ownership of a menu/menuitems are restricted to windows,
screens, and other independent menus.

Only 'independent' structures can be owners, eg. a gadget owned by a
window cannot be an 'owner'.  'Independent' structures must be
explictly freed.

When allocating a structure to an 'owner', if APIG is unable to find
the 'owner' it will allocate the structure as an independent structure
and place it in the appropriate list.

 eg. lets say you load an IFF pic as follows

   pic = loadiff("mypic",itsowner)

   if 'itsowner' is not found by APIG, then pic will be allocated as
   an independent structure and placed in the bitmap list.  You will
   then need to explictly free it with FREEBITMAP(pic).

 eg.
   itext = makeitext(itsowner,...)  /* 'itsowner' not found */

   The IntuiText structure would be placed in the intuitext list,
   and you would then need to use FREEITEXT(itext).
```

1.5 Functions

```
               This node is organised as the original APIG.doc. There is also an
alphabetically sorted list of functions, called:
              Sorted~Functions
               -----------

APIG Library Function Descriptions:

The following functions are provided to allow building and managing of the
basic data structures used by the Intuition, Graphic, and Layer functions.
```

```
GetIDCMP()

GetLayerInfo()

GeTLayerRastPort()

GetRPBitmap()

GetScreenBitmap()

GetScreenRastPort()

GetSTRGad()

GetValue()

GetWindowLayer()

GetWindowRastPort()

GetX()

GetY()

HorizPot()

HorizBody()

IFFDepth()

IFFHeight()

IFFWidth()

IFFViewMode()

IFFColors()

IFFColorTAB()

LoadIFF()

LoadImage()

MakeArea()

MakeBitmap()

MakeBoolGadget()

MakeBorder()

MakeItem()

MakeIText()
```

`DrawEllipse()`

`FindDisplayInfo()`

`Flood()`

`FontExtent()`

`GetDisplayInofData()`

`GetVPModeID()`

`InitArea`
`()`

`InitBitmap`
`()`

`InitRastPort`
`()`

`LoadRGB4()`

`ModeNotAvailable()`

`Move()`

`NextDisplayInfo()`

`OpenFont()`

`OpenMonitor()`

`PolyDraw()`

`ReadPixel()`

`RectFill()`

`ScaleRDiv()`

`ScrollRaster()`

`SetAFPT()`

`SetAPen()`

`SetBPen()`

`SetDRMD()`

`SetDRPT()`

`SetFont()`

`SetOpen()`

`SetRast()`

`SetRGB4()`

`SetSoftStyle()`

`SetWRMSK()`

`Text()`

`TextExtent()`

`TextFit()`

`TextLength()`

`WritePixel()`

`Intuition~Library~Functions`

`ActivateGadget()`

`ActivateWindow()`

`AddGadget()`

`AddGList()`

`AutoRequest()`

`BeginRefresh()`

`BuildEasyRequestArgs()`

`ChangeWindowBox()`

`ClearDMRequest()`

`ClearMenuStrip()`

`ClearPointer()`

`CloseScreen()`

`CloseWindow()`

`DisplayBeep()`

`DisposeObject()`

`DoubleClick()`

`DrawBorder()`

`DrawImage()`

`DrawImageState()`

```
EasyRequest()

EasyRequestArgs()

EndRefresh()

EndRequest()

FreeScreenDrawInfo()

FreeSysRequest()

GetAttr()

GetDefPrefs()

GetDefaultPubScreen()

GetPrefs()

GetScreenData()

GetScreenDrawInfo()

ItemAddress()

ItemNum()

InitRequester~()

IntuiTextLength()

LockPubScreen()

LockPubScreenList()

MenuNum()

ModifyIDCMP()

ModifyProp()

NewModifyProp()

NewObjectA()

NextPubScreen()

MoveScreen()

MoveWindow()

MoveWindowInFrontOf()

OffGadget()
```

`OffMenu()`

`OnGadget()`

`OnMenu()`

`OpenScreenTagList()`

`OpenWindowTagList()`

`PrintIText()`

`PubScreenStatus()`

`RefreshGadgets()`

`RefreshGList()`

`RefreshWindowFrame()`

`RemoveGadget()`

`RemoveGList()`

`ReportMouse()`

`ResetMenuStrip()`

`Request()`

`ScreenToBack()`

`ScreenToFront()`

`SetDefaultPubScreen()`

`SetDMRequest()`

`SetMenuStrip()`

`SetMouseQueue()`

`SetPointer()`

`SetPrefs()`

`SetPubScreenModes()`

`SetWindowTitle()`

`ShowTitle()`

`SizeWindow()`

`SubNum()`

`SysReqHandler()`

```
GT_SetGadgetAttrs()

GT_SetGadgetAttrsA()

LayoutMenuItems()

LayoutMenuItemsA()

LayoutMenus()

LayoutMenusA()

Utility~Library~Functions

AllocateTagItems()

Amiga2Date()

CheckDate()

CloneTagItems()

Date2Amiga()

FilterTagChanges()

FilterTagItems()

FindTagItem()

FreeTagItems()

GetTagData()

MapTags()

NextTagItem()

PackBoolTags()

RefreshTagItemClones()

TagInArray()

Pointer~related

MakeStruct()

MakePointer()
```

1.6 Sorted Functions

```
This is an alphabetically sorted list of all the functions found ↩
   in
```

```
the APIG.Library

------


                ActivateGadget()

                ActivateGadID()

                ActivateWindow()

                Add_List_Node()

                AddGadget()

                AddGList()

                AddHead()

                AddTail()

                AddTo_NewMenu()

                AllocASLRequest()

                AllocateTagItems()

                AllocFileRequest()

                AllocVec()

                Amiga2Date()

                AreaCircle()

                AreADraw()

                AreaEllipse()

                AreaEnd()

                AreaMove()

                ASLRequest()

                AutoRequest()

                BeginRefresh()

                BehindLayer()

                BitmapScale()

                BltBitmap()

                BltBitmapRastPort()
```

`BltClear()`

`BltMaskBitmapRastPort()`

`BltPattern()`

`BltTemplate()`

`BMDepth()`

`BMHeight()`

`BMWidth()`

`BuildEasyRequestArgs()`

`ChangeWindowBox()`

`CheckDate()`

`ClearDMRequest()`

`ClearEOL()`

`ClearMenuStrip()`

`ClearPointer()`

`ClearScreen()`

`ClipBlit()`

`CloneTagItems()`

`CloseFont()`

`CloseMonitor()`

`CloseScreen()`

`CloseWindow()`

`ConvertRawKey()`

`CreateBehindLayer()`

`CreateConText()`

`CreateGadget()`

`CreateGadgetA()`

`CreateMenus()`

`CreateMenusA()`

`CreateUpFrontLayer()`

```
Date2Amiga()

DeleteLayer()

DisplayBeep()

DisposeObject()

DoubleClick()

Draw()

DrawBevelBox()

DrawBorder()

DrawCircle()

DrawEllipse()

DrawImage()

DrawImageState()

EasyRequest()

EasyRequestArgs()

EmptyList()

EndRefresh()

EndRequest()

EnQueue()

FilterTagChanges()

FilterTagItems()

FindDisplayInfo()

FindTagItem()

FirstNode()

Flood()

FontExtent()

Free_Exec_List()

Free_Exec_Node()

FreeArea()
```

FreeASLRequest()

FreeBIRASIM()

FreeBitmap()

FreeFileReq()

FreeGadgets()

FreeImage()

FreeIText()

FreeMenus()

FreeRastPort()

FreeScreenDrawInfo()

FreeSysRequest()

FreeTagItems()

FreeThis()

FreeThisMenu()

FreeVec()

FreeVisualInfo()

GadSelected()

GetArray()

GetAttr()

GetDefaultPubScreen()

GetDefPrefs()

GetDisplayInofData()

GetGadPTR()

GetIDCMP()

GetLayerInfo()

GeTLayerRastPort()

GetPrefs()

GetRPBitmap()

GetScreenBitmap()

GetScreenData()

GetScreenDrawInfo()

GetScreenRastPort()

GetSTRGad()

GetTagData()

GetValue()

GetVisualInfo()

GetVisualInfoA()

GetVPModeID()

GetWindowLayer()

GetWindowRastPort()

GetX()

GetY()

GT_BeginRefresh()

GT_EndRefresh()

GT_RefreshWindow()

GT_SetGadgetAttrs()

GT_SetGadgetAttrsA()

HorizBody()

HorizPot()

IFFColors()

IFFColorTAB()

IFFDepth()

IFFHeight()

IFFViewMode()

IFFWidth()

IMGDepth()

IMGHeight()

IMGWidth()

InitArea
()

InitBitmap
()

InitRastPort
()

InitRequester~()

InsertNode()

IntuiTextLength()

ItemAddress()

ItemNum()

LastNode()

LayoutMenuItems()

LayoutMenuItemsA()

LayoutMenus()

LayoutMenusA()

ListEmpty()

LoadIFF()

LoadImage()

LoadRGB4()

LockPubScreen()

LockPubScreenList()

MakeArea()

MakeBitmap()

MakeBoolGadget()

MakeBorder()

MakeItem()

MakeIText()

MakeMenu()

```
MakeNewGadget()

MakeNewMenu()

MakePointer()

MakePropGadget()

MakeRastPort()

MakeRequester()

MakeStrGadget()

MakeStruct()

MakeStruct()

MakeSubItem()

MakeTAttr()

MapTags()

MenuNum()

MenuNumber()

ModeNotAvailable()

ModifyIDCMP()

ModifyProp()

MouseFrequency()

Move()

MoveLayer()

MoveLayerInFrontOf()

MoveScreen()

MoveWindow()

MoveWindowInFrontOf()

NewList()

NewModifyProp()

NewObjectA()

NextDisplayInfo()

NextPubScreen()
```

`NextTagItem()`

`OffGadget()`

`OffMenu()`

`OnGadget()`

`OnMenu()`

`OpenFont()`

`OpenMonitor()`

`OpenScreen()`

`OpenScreenTagList()`

`OpenWindow()`

`OpenWindowTagList()`

`PackBoolTags()`

`PIText()`

`PolyDraw()`

`PrintIText()`

`PubScreenStatus()`

`ReadPixel()`

`RectFill()`

`RefreshGadgets()`

`RefreshGList()`

`RefreshTagItemClones()`

`RefreshWindowFrame()`

`RemHead()`

`Remove()`

`RemoveGadget()`

`RemoveGList()`

`RemTail()`

`ReportMouse()`

`Request()`

`RequestFile()`

`ResetMenuStrip()`

`SaveIFF()`

`SaveIFFClip()`

`ScaleRDiv()`

`ScreenToBack()`

`ScreenToFront()`

`ScrollLayer()`

`ScrollRaster()`

`ScrProcName()`

`Set_Apig_Globals()`

`SetAFPT()`

`SetAPen()`

`SetArray()`

`SetBPen()`

`SetDefaultPubScreen()`

`SetDMRequest()`

`SetDRMD()`

`SetDRPT()`

`SetFont()`

`SetGadType()`

`SetImage()`

`SetMenuStrip()`

`SetMouseQueue()`

`SetNewGadget()`

`SetOpen()`

`SetPointer()`

`SetPrefs()`

`SetPubScreenModes()`

`SetRast()`

`SetRGB4()`

`SetSelect()`

`SetSoftStyle()`

`SetStrGad()`

`SetStrGadID()`

`SetTagSlot()`

`SetValue()`

`SetWindowTitle()`

`SetWRMSK()`

`SetX()`

`SetY()`

`ShowTitle()`

`SizeLayer()`

`SizeWindow()`

`SubNum()`

`SysReqHandler()`

`TagInArray()`

`Text()`

`TextExtent()`

`TextFit()`

`TextLength()`

`TickFrequency()`

`UnlockPubScreen()`

`UpFrontLayer()`

`UseIFFColor()`

`VertBody()`

```
                VertPot()

                ViewAddress()

                ViewPortAddress()

                WbenchToBack()

                WbenchToFront()

                WindowInfo()

                WindowLimits()

                WindowToBack()

                WindowToFront()

                WinTaskName()

                WriteConsole()

                WritePixel()

                ZipWindow()
```

1.7 Function reference

```
                 This node is organised as the original APIG.doc. There is also an
alphabetically sorted list of functions, called:
                Sorted~Function reference
                 -----------

APIG Library Function Descriptions:

The following functions are provided to allow building and managing of the
basic data structures used by the Intuition, Graphic, and Layer functions.



                ActivateGadID(gadgetid,window,requester)

                Add_List_Node(listptr,~string,~position,~nodesize,~pri,~type)

                AddTo_NewMenu(apignmdata,type,label,commkey,flags,mutual,usrdata)

                ConvertRawKey(keycode,qualifier,keymap)

                BMDepth(bm)

                BMHeight(bm)

                BMWidth(bm)

                IMGDepth(image)
```

```
IMGHeight(image)

IMGWidth(image)

FreeArea(window)

FreeBitmap(pointertobitmap)

FreeImage(pointertoimage)

FreeRastPort(pointertorastport)

FreeBIRASIM(pointer)

FreeThisMenu(menustrippointer)

FreeIText(intuitextpointer)

FreeThis(pointer~to~any~independent~structure)

Free_Exec_List(listptr,~nodestructsize,~liststructsize)

Free_Exec_Node(nodeptr,~nodesize)

GadSelected(gadgetptr)

GetArray(arrayptr,arrayindx)

GetGadPTR(window,gadgetid,requester)

GetIDCMP(window)

GetLayerInfo(layer)

GeTLayerRastPort(layer)

GetRPBitmap(rp)

GetScreenBitmap(screen)

GetScreenRastPort(screen)

GetSTRGad(window,gadgetid,requester)

GetValue(ptr,offset,size,type)

GetWindowLayer(window)

GetWindowRastPort(window)

GetX(arrayptr,xindex)

GetY(arrayptr,yindex)

HorizPot(propgadgetptr)
```

```
HorizBody(propgadgetptr)

IFFDepth(pointer)

IFFHeight(pointer)

IFFWidth(pointer)

IFFViewMode(pointer)

IFFColors(pointer)

IFFColorTAB(pointer)

LoadIFF(filename,owner)

LoadImage(filename,imageptr,left,top,owner)

MakeArea(window,xsize,ysize,maxvectors)

MakeBitmap(width,height,depth,owner)

MakeBoolGadget(owner,left,top,width,hgt,flags,activation,itext, ←
   bpen,render,select,gadid,linkto)

MakeBorder(owner,arrayptr,arraycnt,left,top,fp,bp,dm,linkto)

MakeItem(menustrip,text,menu,left,top,width,height,flags,ME,COM,fp ←
   ,bp,dm,itemfill,selectfill)

MakeIText(owner,text,xpos,ypos,fpen,bpen,dmode,fontattr,linkto)

MakeMenu(menuowner,menutext,leftedge,width,flags,menupointer)

MakeNewGadget(vinfo,font,left,top,width,height,text,flags,id, ←
   usrdata)

MakeNewMenu(n)

MakePropGadget(owner,left,top,width,hgt,flags,activation,itext,, ←
   piflags,hbody,vbody,gadid,linkto

,knobimage)

MakeRastPort(width,height,depth,owner)

MakeRequester(window,left,top,width,height,gadget,text,border, ←
   backfill,flags,relleft,reltop,bm)

MakeStrGadget(window,left,top,width,hgt,flags,activation,itext, ←
   bpen,render,select,gadid,linkto,s

trlen,undobuf)

MakeSubItem(menustrip,text,item,left,top,width,height,flags,ME,COM ←
   ,fp,bp,dm,itemfill,selectfill)
```

```
MakeStruct(owner,type,size,mem_type)

MakeTAttr(window,fontname,fontsize)

MenuNumber(menustrip,menu,item,subitem)

MouseFrequency(window,N)

OpenWindow(portname,left,top,wid,hgt,dpen,bpen,IDCMP,flags,title, ↩
    scr,console,bitmap,chkmark,gadl

ist)

OpenScreen(left,top,width,height,depth,dpen,bpen,vmodes,type,title ↩
    )

PIText(rp,left,top,text,fp,bp,dm,font)

SaveIFF(bitmap,filename,colortab,HAM,compress)

SaveIFFClip(bitmap,filename,x,y,w,h,colortab,HAM,compress)

ScrProcName(scrptr)

Set_Apig_Globals()

SetArray(arrayptr,~arrayindx,~value)

SetNewGadget(ngad,vinfo,font,left,top,width,height,text,flags,id, ↩
    usrdata)

SetSelect(gadgetptr,state)

SetGadType(gadgetptr,gadtype)

SetImage(image,left,top,ppick,ponoff)

SetStrGad(gadgetptr,text)

SetStrGadID(window,gadid,text,requester)

SetTagSlot(tagarray,slot,tag,'p'/'n',value)

SetValue(ptr,offset,size,type,value,len)

SetX(arrayptr,xindex,value)

SetY(arrayptr,yindex,value)

VertPot(propgadgeptr)

VertBody(propgadgetptr)

TickFrequency(window,N)

UseIFFColor(pointer,scr)
```

```
WindowInfo(window,code)

WinTaskName(window)

WriteConsole(window,text)

Graphics~Library~Functions

AreaCircle(rp,cx,cy,radius)

AreADraw(rp,x,y)

AreaEllipse(rp,cx,cy,a,b)

AreaEnd(rp)

AreaMove(rp,x,y)

BitmapScale(bitscaleargs)

BltBitmap(srcbm,srcx,srcy,dstbm,dstx,dsty,sizex,sizey,minterm,mask ↩
    ,tempa)

BltBitmapRastPort(srcbm,srcx,srcy,rp,destx,desty,sizex,sizey, ↩
    minterm)

BltClear(memblock,bytecount,flags)

BltMaskBitmapRastPort(scrbm,srcx,srcy,rp,destx,desty,sizex,sizey, ↩
    minterm,bltmask)

BltPattern(rp,mask,x1,y1,x2,y2,bytecnt)

BltTemplate(srctemplate,srcx,srcmod,rp,dstx,dsty,sizex,sizey)

ClearEOL(rp,x,y)

ClearScreen(rp,x,y)

ClipBlit(srcrp,srcx,srcy,destrp,destx,desty,xsize,ysize,minterm)

CloseFont(font)

CloseMonitor(monitor_spec)

Draw(rp,x,y)

DrawCircle(rp,cx,cy,r)

DrawEllipse(rp,cx,cy,a,b)

FindDisplayInfo(id)

Flood(rp,mode,x,y)

FontExtent(font,fontextent)
```

```
GetDisplayInofData(handle,buf,size,tag,id)

GetVPModeID(viewport)

InitArea
 ()

InitBitmap
 ()

InitRastPort
 ()

LoadRGB4(screen,arrayptr,count)

ModeNotAvailable(id)

Move(rp,x,y)

NextDisplayInfo(id)

OpenFont(textAttr)

OpenMonitor(monitor_name,display_id)

PolyDraw(rp,count,array)

ReadPixel(rp,x,y)

RectFill(rp,xmin,ymin,xmax,ymax)

ScaleRDiv(factor,numerator,denominator)

ScrollRaster(rp,dx,dy,xmin,ymin,xmax,ymax)

SetAFPT(rp,pattern,patternsize)

SetAPen(rp,pen)

SetBPen(rp,pen)

SetDRMD(rp,mode)

SetDRPT(rp,linepattern)

SetFont(rp,font)

SetOpen(rp,pen)

SetRast(rp,pen)

SetRGB4(screen/window,pen,r,g,b)

SetSoftStyle(rp,style,enable)

SetWRMSK(rp,wrtmask)
```

```
Text(rp,string,count)

TextExtent(rp,string,count,textextent)

TextFit(rp,string,len,textextent,consextent,strdir,consbitwid, ↩
   consbithgt)

TextLength(rp,string,count)

WritePixel(rp,x,y)

Intuition~Library~Functions

ActivateGadget(gadget,window,requester)

ActivateWindow(window)

AddGadget(window,gadget,position)

AddGList(window,gadget,position,numgad,requester)

AutoRequest(window,itext,itext,itext,posflags,newflags,width,hgt)

BeginRefresh(window)

BuildEasyRequestArgs(window,easystruct,idcmp,args)

ChangeWindowBox(window,left,top,width,height)

ClearDMRequest(window)

ClearMenuStrip(window)

ClearPointer(window)

CloseScreen(screen)

CloseWindow(window)

DisplayBeep(screen)

DisposeObject(object)

DoubleClick(startsecs,startmicros,currentsecs,currentmicros)

DrawBorder(rp,border,leftoffset,topoffset)

DrawImage(rp,image,leftoffset,topoffset)

DrawImageState(rp,image,leftoffset,topoffset,state,drinfo)

EasyRequest(window,title,bodytext,gadtext,arglist,IDCMP,flags)

EasyRequestArgs(window,es,IDCMP_ptr,ArgList)

EndRefresh(window,complete)
```

```
EndRequest(requester,window)

FreeScreenDrawInfo(drinfo)

FreeSysRequest(window)

GetAttr(attrid,object,storageptr)

GetDefPrefs(prefbuffer,size)

GetDefaultPubScreen(namebuffer)

GetPrefs(prefbuffer,size)

GetScreenData(buffer,size,type,screen)

GetScreenDrawInfo(screen)

ItemAddress(menustrip,menunumber)

ItemNum(menunumber)

InitRequester~()

IntuiTextLength(itext)

LockPubScreen(screename/null)

LockPubScreenList(screename/null)

MenuNum(menunumber)

ModifyIDCMP(window,idcmpflags)

ModifyProp(gadget,window,requester,flags,hpot,vpot,hbody,vbody)

NewModifyProp(gadget,window,requester,flags,hpot,vpot,hbody,vbody, ↩
   numgad)

NewObjectA(class,classid,taglist)

NextPubScreen(screen,namebuffer)

MoveScreen(screen,deltax,deltay)

MoveWindow(window,deltax,deltay)

MoveWindowInFrontOf(window,behindwindow)

OffGadget(gadget,window,requester)

OffMenu(window,menunumber)

OnGadget(gadget,window,requester)

OnMenu(window,menunumber)
```

```
OpenScreenTagList(newscreen,taglist)

OpenWindowTagList(portname,newwindow,taglist,console)

PrintIText(rp,itext,leftoffset,topoffset)

PubScreenStatus(screen,statusflags)

RefreshGadgets(gadgets,window,requester)

RefreshGList(gadgets,window,requester,numgad)

RefreshWindowFrame(window)

RemoveGadget(window,gadget)

RemoveGList(window,gadget,numgad)

ReportMouse(boolean,window)

ResetMenuStrip(window,menu)

Request(requester,window)

ScreenToBack(screen)

ScreenToFront(screen)

SetDefaultPubScreen()

SetDMRequest(window,dmrequester)

SetMenuStrip(window,menu)

SetMouseQueue(window,newlength)

SetPointer(window,pointer,height,width,xoffset,yoffset)

SetPrefs(prefbuffer,size,inform)

SetPubScreenModes(modes)

SetWindowTitle(window,widnowtitle,screentitle)

ShowTitle(screen,showit)

SizeWindow(window,deltax,deltay)

SubNum(menunumber)

SysReqHandler(window,idcmpflagsptr,waitinput)

UnlockPubScreen(screenname/null,screenptr)

ViewAddress()

ViewPortAddress(window)
```

```
WbenchToBack()

WbenchToFront()

WindowLimits(window,minwidth,minheight,maxwidth,maxheight)

WindowToBack(window)

WindowToFront(window)

ZipWindow(window)

Layers~Library~Functions

BehindLayer(layer)

CreateBehindLayer(windowpointer,x0,y0,x1,y1,flags,bm2)

CreateUpFrontLayer(windowpointer,x0,y0,x1,y1,flags,bm2)

DeleteLayer(layer)

MoveLayerInFrontOf(layertomove,targetlayer)

MoveLayer(layer,dx,dy)

ScrollLayer(layer,dx,dy)

SizeLayer(layer,dx,dy)

UpFrontLayer(layer)

Exec~Library~Functions

AddHead(list,node)

AddTail(list,node)

AllocVec(size,type)

EnQueue(list,node)

FreeVec(vecptr)

InsertNode(list,node,listnode)

NewList(list)

RemHead(list)

Remove(node)

RemTail(list)

Apig~Library~List~related~functions
```

```
EmptyList(list)

ListEmpty(list)

FirstNode(list,node)

LastNode(list,node)

ASL~Library~Functions

AllocASLRequest(type,tags)

AllocFileRequest()

ASLRequest(req,tags,owner)

FreeASLRequest(filerequest)

FreeFileReq(filerequest)

RequestFile(req,multi,save,hail,dir,file,pat,nofile,win,left,top, ↩
   width,hgt,sep)

Gadtools~Library~Functions

CreateConText(&gadptr~)

CreateGadget(kind,previous,newgad,tag1,tag1data,...)

CreateGadgetA(kind,previous,newgad,taglist)

CreateMenus(newmenu,tag1,tag1data,...)

CreateMenusA(newmenu,taglist)

DrawBevelBox(rp,l,t,w,h,vi,GTBB_Recessed)

FreeGadgets(glist)

FreeMenus(menu)

FreeVisualInfo(vi)

GetVisualInfo(scr,tag1,tag1data,...)

GetVisualInfoA(scr,taglist)

GT_BeginRefresh(window)

GT_EndRefresh(window,TRUE/FALSE)

GT_RefreshWindow(window)

GT_SetGadgetAttrs(gad,window,requester,tags)

GT_SetGadgetAttrsA(gad,window,requester,taglist)
```

```
LayoutMenuItems(menuitem,vi,tag1,tag1data,...)

LayoutMenuItemsA(menuitem,vi,taglist)

LayoutMenus(menu,vi,tag1,tag1data,...)

LayoutMenusA(menu,vi,taglist)

Utility~Library~Functions

AllocateTagItems(N)

Amiga2Date(amigatime,date)

CheckDate(date)

CloneTagItems(taglist)

Date2Amiga(date)

FilterTagChanges(changelist,oldvalues,apply)

FilterTagItems(taglist,tagarray,logic)

FindTagItem(tagval,taglist)

FreeTagItems(taglist)

GetTagData(tagval,default,taglist)

MapTags(taglist,maplist,includemiss)

NextTagItem(tagitemptr)

PackBoolTags(intialflags,taglist,boolmap)

RefreshTagItemClones(clonetagitems,originaltagitems)

TagInArray(tag,tagarray)

Pointer~related

MakeStruct(owner,type,size,memtype)

MakePointer()
```

1.8 Sorted function reference

```
 This node is sorted by the alphabet.

-----------


ActivateGadget(gadget,window,requester)
```

```
ActivateGadID(gadgetid,window,requester)

ActivateWindow(window)

Add_List_Node(listptr,~string,~position,~nodesize,~pri,~type)

AddGadget(window,gadget,position)

AddGList(window,gadget,position,numgad,requester)

AddHead(list,node)

AddTail(list,node)

AddTo_NewMenu(apignmdata,type,label,commkey,flags,mutual,usrdata)

AllocASLRequest(type,tags)

AllocateTagItems(N)

AllocFileRequest()

AllocVec(size,type)

Amiga2Date(amigatime,date)

AreaCircle(rp,cx,cy,radius)

AreADraw(rp,x,y)

AreaEllipse(rp,cx,cy,a,b)

AreaEnd(rp)

AreaMove(rp,x,y)

ASLRequest(req,tags,owner)

AutoRequest(window,itext,itext,itext,posflags,newflags,width,hgt)

BeginRefresh(window)

BehindLayer(layer)

BitmapScale(bitscaleargs)

BltBitmap(srcbm,srcx,srcy,dstbm,dstx,dsty,sizex,sizey,minterm,mask ↩
    ,tempa)

BltBitmapRastPort(srcbm,srcx,srcy,rp,destx,desty,sizex,sizey, ↩
    minterm)

BltClear(memblock,bytecount,flags)

BltMaskBitmapRastPort(scrbm,srcx,srcy,rp,destx,desty,sizex,sizey, ↩
    minterm,bltmask)
```

```
BltPattern(rp,mask,x1,y1,x2,y2,bytecnt)

BltTemplate(srctemplate,srcx,srcmod,rp,dstx,dsty,sizex,sizey)

BMDepth(bm)

BMHeight(bm)

BMWidth(bm)

BuildEasyRequestArgs(window,easystruct,idcmp,args)

ChangeWindowBox(window,left,top,width,height)

CheckDate(date)

ClearDMRequest(window)

ClearEOL(rp,x,y)

ClearMenuStrip(window)

ClearPointer(window)

ClearScreen(rp,x,y)

ClipBlit(srcrp,srcx,srcy,destrp,destx,desty,xsize,ysize,minterm)

CloneTagItems(taglist)

CloseFont(font)

CloseMonitor(monitor_spec)

CloseScreen(screen)

CloseWindow(window)

ConvertRawKey(keycode,qualifier,keymap)

CreateBehindLayer(windowpointer,x0,y0,x1,y1,flags,bm2)

CreateConText(&gadptr~)

CreateGadget(kind,previous,newgad,tag1,tag1data,...)

CreateGadgetA(kind,previous,newgad,taglist)

CreateMenus(newmenu,tag1,tag1data,...)

CreateMenusA(newmenu,taglist)

CreateUpFrontLayer(windowpointer,x0,y0,x1,y1,flags,bm2)

Date2Amiga(date)
```

```
DeleteLayer(layer)

DisplayBeep(screen)

DisposeObject(object)

DoubleClick(startsecs,startmicros,currentsecs,currentmicros)

Draw(rp,x,y)

DrawBevelBox(rp,l,t,w,h,vi,GTBB_Recessed)

DrawBorder(rp,border,leftoffset,topoffset)

DrawCircle(rp,cx,cy,r)

DrawEllipse(rp,cx,cy,a,b)

DrawImage(rp,image,leftoffset,topoffset)

DrawImageState(rp,image,leftoffset,topoffset,state,drinfo)

EasyRequest(window,title,bodytext,gadtext,arglist,IDCMP,flags)

EasyRequestArgs(window,es,IDCMP_ptr,ArgList)

EmptyList(list)

EndRefresh(window,complete)

EndRequest(requester,window)

EnQueue(list,node)

FilterTagChanges(changelist,oldvalues,apply)

FilterTagItems(taglist,tagarray,logic)

FindDisplayInfo(id)

FindTagItem(tagval,taglist)

FirstNode(list,node)

Flood(rp,mode,x,y)

FontExtent(font,fontextent)

Free_Exec_List(listptr,~nodestructsize,~liststructsize)

Free_Exec_Node(nodeptr,~nodesize)

FreeArea(window)

FreeASLRequest(filerequest)

FreeBIRASIM(pointer)
```

```
FreeBitmap(pointertobitmap)

FreeFileReq(filerequest)

FreeGadgets(glist)

FreeImage(pointertoimage)

FreeIText(intuitextpointer)

FreeMenus(menu)

FreeRastPort(pointertorastport)

FreeScreenDrawInfo(drinfo)

FreeSysRequest(window)

FreeTagItems(taglist)

FreeThis(pointer~to~any~independent~structure)

FreeThisMenu(menustrippointer)

FreeVec(vecptr)

FreeVisualInfo(vi)

GadSelected(gadgetptr)

GetArray(arrayptr,arrayindx)

GetAttr(attrid,object,storageptr)

GetDefaultPubScreen(namebuffer)

GetDefPrefs(prefbuffer,size)

GetDisplayInofData(handle,buf,size,tag,id)

GetGadPTR(window,gadgetid,requester)

GetIDCMP(window)

GetLayerInfo(layer)

GeTLayerRastPort(layer)

GetPrefs(prefbuffer,size)

GetRPBitmap(rp)

GetScreenBitmap(screen)

GetScreenData(buffer,size,type,screen)
```

```
GetScreenDrawInfo(screen)

GetScreenRastPort(screen)

GetSTRGad(window,gadgetid,requester)

GetTagData(tagval,default,taglist)

GetValue(ptr,offset,size,type)

GetVisualInfo(scr,tag1,tag1data,...)

GetVisualInfoA(scr,taglist)

GetVPModeID(viewport)

GetWindowLayer(window)

GetWindowRastPort(window)

GetX(arrayptr,xindex)

GetY(arrayptr,yindex)

GT_BeginRefresh(window)

GT_EndRefresh(window,TRUE/FALSE)

GT_RefreshWindow(window)

GT_SetGadgetAttrs(gad,window,requester,tags)

GT_SetGadgetAttrsA(gad,window,requester,taglist)

HorizBody(propgadgetptr)

HorizPot(propgadgetptr)

IFFColors(pointer)

IFFColorTAB(pointer)

IFFDepth(pointer)

IFFHeight(pointer)

IFFViewMode(pointer)

IFFWidth(pointer)

IMGDepth(image)

IMGHeight(image)

IMGWidth(image)

InitArea
```

```
 ()

InitBitmap
 ()

InitRastPort
 ()

InitRequester~()

InsertNode(list,node,listnode)

IntuiTextLength(itext)

ItemAddress(menustrip,menunumber)

ItemNum(menunumber)

LastNode(list,node)

LayoutMenuItems(menuitem,vi,tag1,tag1data,...)

LayoutMenuItemsA(menuitem,vi,taglist)

LayoutMenus(menu,vi,tag1,tag1data,...)

LayoutMenusA(menu,vi,taglist)

ListEmpty(list)

LoadIFF(filename,owner)

LoadImage(filename,imageptr,left,top,owner)

LoadRGB4(screen,arrayptr,count)

LockPubScreen(screename/null)

LockPubScreenList(screename/null)

MakeArea(window,xsize,ysize,maxvectors)

MakeBitmap(width,height,depth,owner)

MakeBoolGadget(owner,left,top,width,hgt,flags,activation,itext, ↩
    bpen,render,select,gadid,linkto)

MakeBorder(owner,arrayptr,arraycnt,left,top,fp,bp,dm,linkto)

MakeItem(menustrip,text,menu,left,top,width,height,flags,ME,COM,fp ↩
    ,bp,dm,itemfill,selectfill)

MakeIText(owner,text,xpos,ypos,fpen,bpen,dmode,fontattr,linkto)

MakeMenu(menuowner,menutext,leftedge,width,flags,menupointer)
```

```
MakeNewGadget(vinfo,font,left,top,width,height,text,flags,id, ↩
   usrdata)

MakeNewMenu(n)

MakePointer()

MakePropGadget(owner,left,top,width,hgt,flags,activation,itext,, ↩
   piflags,hbody,vbody,gadid,linkto,kn

obimage)

MakeRastPort(width,height,depth,owner)

MakeRequester(window,left,top,width,height,gadget,text,border, ↩
   backfill,flags,relleft,reltop,bm)

MakeStrGadget(window,left,top,width,hgt,flags,activation,itext, ↩
   bpen,render,select,gadid,linkto,strl

en,undobuf)

MakeStruct(owner,type,size,mem_type)

MakeStruct(owner,type,size,memtype)

MakeSubItem(menustrip,text,item,left,top,width,height,flags,ME,COM ↩
   ,fp,bp,dm,itemfill,selectfill)

MakeTAttr(window,fontname,fontsize)

MapTags(taglist,maplist,includemiss)

MenuNum(menunumber)

MenuNumber(menustrip,menu,item,subitem)

ModeNotAvailable(id)

ModifyIDCMP(window,idcmpflags)

ModifyProp(gadget,window,requester,flags,hpot,vpot,hbody,vbody)

MouseFrequency(window,N)

Move(rp,x,y)

MoveLayer(layer,dx,dy)

MoveLayerInFrontOf(layertomove,targetlayer)

MoveScreen(screen,deltax,deltay)

MoveWindow(window,deltax,deltay)

MoveWindowInFrontOf(window,behindwindow)
```

```
NewList(list)

NewModifyProp(gadget,window,requester,flags,hpot,vpot,hbody,vbody, ↩
    numgad)

NewObjectA(class,classid,taglist)

NextDisplayInfo(id)

NextPubScreen(screen,namebuffer)

NextTagItem(tagitemptr)

OffGadget(gadget,window,requester)

OffMenu(window,menunumber)

OnGadget(gadget,window,requester)

OnMenu(window,menunumber)

OpenFont(textAttr)

OpenMonitor(monitor_name,display_id)

OpenScreen(left,top,width,height,depth,dpen,bpen,vmodes,type,title ↩
    )

OpenScreenTagList(newscreen,taglist)

OpenWindow(portname,left,top,wid,hgt,dpen,bpen,IDCMP,flags,title, ↩
    scr,console,bitmap,chkmark,gadlist

)

OpenWindowTagList(portname,newwindow,taglist,console)

PackBoolTags(intialflags,taglist,boolmap)

PIText(rp,left,top,text,fp,bp,dm,font)

PolyDraw(rp,count,array)

PrintIText(rp,itext,leftoffset,topoffset)

PubScreenStatus(screen,statusflags)

ReadPixel(rp,x,y)

RectFill(rp,xmin,ymin,xmax,ymax)

RefreshGadgets(gadgets,window,requester)

RefreshGList(gadgets,window,requester,numgad)

RefreshTagItemClones(clonetagitems,originaltagitems)
```

```
RefreshWindowFrame(window)

RemHead(list)

Remove(node)

RemoveGadget(window,gadget)

RemoveGList(window,gadget,numgad)

RemTail(list)

ReportMouse(boolean,window)

Request(requester,window)

RequestFile(req,multi,save,hail,dir,file,pat,nofile,win,left,top, ↩
   width,hgt,sep)

ResetMenuStrip(window,menu)

SaveIFF(bitmap,filename,colortab,HAM,compress)

SaveIFFClip(bitmap,filename,x,y,w,h,colortab,HAM,compress)

ScaleRDiv(factor,numerator,denominator)

ScreenToBack(screen)

ScreenToFront(screen)

ScrollLayer(layer,dx,dy)

ScrollRaster(rp,dx,dy,xmin,ymin,xmax,ymax)

ScrProcName(scrptr)

Set_Apig_Globals()

SetAFPT(rp,pattern,patternsize)

SetAPen(rp,pen)

SetArray(arrayptr,~arrayindx,~value)

SetBPen(rp,pen)

SetDefaultPubScreen()

SetDMRequest(window,dmrequester)

SetDRMD(rp,mode)

SetDRPT(rp,linepattern)

SetFont(rp,font)
```

```
SetGadType(gadgetptr,gadtype)

SetImage(image,left,top,ppick,ponoff)

SetMenuStrip(window,menu)

SetMouseQueue(window,newlength)

SetNewGadget(ngad,vinfo,font,left,top,width,height,text,flags,id, ↩
    usrdata)

SetOpen(rp,pen)

SetPointer(window,pointer,height,width,xoffset,yoffset)

SetPrefs(prefbuffer,size,inform)

SetPubScreenModes(modes)

SetRast(rp,pen)

SetRGB4(screen/window,pen,r,g,b)

SetSelect(gadgetptr,state)

SetSoftStyle(rp,style,enable)

SetStrGad(gadgetptr,text)

SetStrGadID(window,gadid,text,requester)

SetTagSlot(tagarray,slot,tag,’p’/’n’,value)

SetValue(ptr,offset,size,type,value,len)

SetWindowTitle(window,widnowtitle,screentitle)

SetWRMSK(rp,wrtmask)

SetX(arrayptr,xindex,value)

SetY(arrayptr,yindex,value)

ShowTitle(screen,showit)

SizeLayer(layer,dx,dy)

SizeWindow(window,deltax,deltay)

SubNum(menunumber)

SysReqHandler(window,idcmpflagsptr,waitinput)

TagInArray(tag,tagarray)

Text(rp,string,count)
```

```
TextExtent(rp,string,count,textextent)

TextFit(rp,string,len,textextent,consextent,strdir,consbitwid, ←
   consbithgt)

TextLength(rp,string,count)

TickFrequency(window,N)

UnlockPubScreen(screenname/null,screenptr)

UpFrontLayer(layer)

UseIFFColor(pointer,scr)

VertBody(propgadgetptr)

VertPot(propgadgeptr)

ViewAddress()

ViewPortAddress(window)

WbenchToBack()

WbenchToFront()

WindowInfo(window,code)

WindowLimits(window,minwidth,minheight,maxwidth,maxheight)

WindowToBack(window)

WindowToFront(window)

WinTaskName(window)

WriteConsole(window,text)

WritePixel(rp,x,y)

ZipWindow(window)
```

1.9 Error codes

```
   APIG error codes

 5001    ; window pointer not valid
 5002    ; screen pointer not valid
 5003    ; gadget pointer not valid
 5004    ; requester pointer not valid
 5005    ; intuitext pointer not valid
 5006    ; menu pointer not valid
 5007    ; menuitem pointer not valid
 5008    ; subitem pointer not valid
```

```
  5009    ; iff pointer not valid
  5010    ; expected non-null pointer
  5011    ; expected non-zero numeric
  5012    ; could not open iff.library
  5013    ; no console attached to window
  5014    ; invalid size specified in getvalue/setvalue
  5015    ; invalid owner specified for menu/item/subitem
  5016    ; could not load font
  5017    ; could not load iff
  5018    ; border points  > 127 or thick > 25
  5019    ; invalid "link to" pointer
  5080    ; screenopen failed
  5081    ; attempt to closescreen with a window open
  5090    ; port not found
  5095    ; memory allocation failure
  5098    ; attempt to access low mem
  5099    ; task creation failure
  5200    ; not using dos v37 2.04
  5201    ; not a asl file requester pointer
  5202    ; asl file allocation failed
  5203    ; GadTool gadget creation failure
  5204    ; boopsi class object pointer invalid
```

1.10 Translators node

Written on: Friday, 13 October, 1995

I wanted to start using APIG.library after I had read some Arexx tutorials where it was used. But seing that there was just an oldfashion doc-file for it, I started 'translating it into an AmigaGuide-doc.

I have basically just split the original doc up in some nodes and linked them together. I have made two nodes containing function-names, one which is in the same order as they where in the original Apig.doc, and one where all the functions have been sorted by the alphabet.

I also made a reference-node, which is actually just the same as the the node with the unsorted Apig-functions, but this one also contains all the parameters for the functions. This is a usefull node to have when you remember what the function is called, but not the exact parameters for it. This node is also available, sorted by the alphabet.

It might seem strange to make 4 nodes which actually references the same functions, but when you are new to the Apig.library you will probably like the nodes with the functions names, which does not include their parameters, since, in my opinion, they are easier to read, and since you're new, you probably needs to read the full describtion for the function anyway. Then when you get more experienced, you will probably be glad that there is a node which contains all function-names and thier parameters. This way, you can easily find the parameters for the function you are about to use, in case you have forgot them. But you can still look at the complete describtion of the function, should you like to.

They are organised, as said, one by alphabet, and one as the original Apig.doc. If you know the name of the function you are looking for, you

```
probably use the alphabetically sorted node, but if you just know what kind
of function you need, you look in the Apig-formatted node.

 I have changed all function names, so they no longer are only upper-case, but
mixed, sort of like in the RKM-books. (This was done for readability.)

 My native language is danish, (Ain't that something you eat?), so please
excuse any spelling mistakes, and wrong ways of expressing myself.

 If you use this guide, please send me a postcard, a letter or something.


 - Mads

 -----------

 My address:

   Mads Lie Jensen
   Tværvangen 23
   Rakkeby
   DK-9800 Hjørring
   Denmark
```

1.11 ActivataGadID

```
 <>  ACTIVATEGADID(gadgetid,window,requester)

      Tries to activate the gadget with ID 'gadgetid' in the window.

      Inputs:

          gadgetid   - numeric, gadget's ID

          window     - pointer, gadgets window

          requester  - pointer, pointer to requester if in a requester
                       else make this null.

     Returns:        - returns TRUE if activated otherwise zero.
```

1.12 ADD_LIST_NODE

```
 <>  ADD_LIST_NODE(listptr, string, position, nodesize, pri, type)

    Builds an Exec Node structure and inserts it into the supplied
    list.  The nodes LN_NAME parameter will point to 'string'.
    Intended for building list structures for use with LISTVIEW_KIND
    gadgets, but you probably will find other uses for it.

          listptr    - pointer, hex-string pointer to List structure,
                       which has been properly initialized with NEWLIST().
                       If null, '0000 0000'x, then no insertion of any kind
                       is done, but the allocated node pointer is returned.

          string     - string, literal/variable which will be
                       used for nodes LN_NAME parameter.

          position   - numeric, determines where to insert the node as
                       follows:
                       if position = -1, enqueue using node priority
                       if position = 0, add to tail of list (default)
                       if position = 1, add to head of list
                       if position > 1 or < -1 , insert at abs('position')
                       eg. add_list_node(list,string,3), makes the new
                           node the third node in the list
                       This parameter is optional, default value is 0,
                       will add to tail of list.

          nodesize   - numeric, the amount of memory to be allocated to
                       the node. This parameter is optional if not
                       specified or if less than 14, then it defaults
                       to 14 (the size of an ExecList node).  This allows
                       you to allocate room for additonal data to be
                       stored with each node using SETVALUE().

          pri        - numeric, value to set nodes LN_PRI parameter.
                       This parameter is optional, default value is 0.

          type       - numeric, value to set nodes LN_TYPE parameter.
                       This parameter is optional, default value is 0.

     Returns:        - pointer to allocated Node structure, returns null
                       if failure occurs.

     Note: The list can be manipulated with the Exec List functions
           Remove()/AddTail()/etc. If you are gonna manipulate this
           list then you should know that all memory is allocated with
           the Exec ALLOCMEM function. The LN_NAME parameter points to
           a block of memory which is null terminated.
           ie. length(LN_NAME)+1 is used in FREEMEM().
           APIG does NOT keep track of the node memory allocated with
           this function, ie. FREETHIS() will not free it.
           If you permanently remove a node from the list then
           you should free up the memory with FREE_EXEC_NODE().

      See Also FREE_EXEC_LIST,FREE_EXEC_NODE.
```

1.13 ADDTO_NEWMENU

```
 <>  ADDTO_NEWMENU(apignmdata,type,label,commkey,flags,mutual,usrdata)

      This function allows you to dynamically build a menu structure for
      use with the GADTools CREATEMENU() functions.

      Inputs:
```

```
apignmdata - pointer, this MUST be the returned value from
             MAKENEWMENU() function.  This is an internal
             APIG structure used to build the NewMenu struct.
             ADDTO_NEWMENU() will re-allocate the NewMenu
             structure as it grows, thus you do not need to
             know before hand how many items will be in your
             menu, just add them in.

type       - numeric, type of menu item
             eg. NM_TITLE/NM_ITEM/NM_SUB/NM_END

label      - string, text for menu header/item/subitem
             you may also specify NM_BARLABEL here as well

commkey    - string, single char for R-AMIGA command sequence char
             Use "" (empty string) for no commkey.

flags      - numeric, menu or menuitem flags

mutual     - numeric, mutual exclusion mask

usrdata    - pointer, to what ever you want to point to

Returns    - always returns 1

Note to use this function you must first call MAKENEWMENU() and
use the return value as the first parameter in ADDTO_NEWMENU().

Then you can build your menu as follows:

apignmdata = MAKENEWMENU()
call ADDTO_NEWMENU(apignmdata,NM_TITLE,"Project",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_ITEM,"New",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_ITEM,"Open",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_ITEM,"Close",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_ITEM,"Save",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_SUB,"Save As New",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_SUB,"Save as Old",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_ITEM,NM_BARLABEL,0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_ITEM,"Quit",0,0,0,null())
call ADDTO_NEWMENU(apignmdata,NM_END,"",0,0,0,null())

In 'C' it would be:
struct NewMenu mymenu[] = {
         NM_TITLE,"Project",0,0,0,NULL,
         NM_ITEM,"New",0,0,0,NULL,
         NM_ITEM,"Open",0,0,0,NULL,
         NM_ITEM,"Close",0,0,0,NULL,
         NM_ITEM,"Save",0,0,0,NULL,
         NM_SUB,"Save As New",0,0,0,NULL,
         NM_SUB,"Save as Old",0,0,0,NULL,
         NM_ITEM,NM_BARLABEL,0,0,0,NULL,
         NM_ITEM,"Quit",0,0,0,NULL,
         NM_END,0,0,0,0,NULL                };

 See Also MAKENEWMENU()
```

1.14 CONVERTRAWKEY

```
 <>  CONVERTRAWKEY(keycode,qualifier,keymap)

      This function is used to convert a 'raw' keycode value into
      its equivalent ascii character string.

      Inputs:

         keycode    - numeric, the raw keycode value to be converted

         qualifier - numeric, the value of any key qualifiers such as
                     LSHIFT,LAMIGA, etc.

         keymap     - pointer to a keymap structure
                     This value is currently ignored, but must be
                     present.  In a future release keymap support
                     maybe provided, for now code this as zero.

         Returns    - the ascii character string for the specified keycode
                     and key qualifier.
```

1.15 BM

```
 <>  BMDEPTH(bm)
 <>  BMHEIGHT(bm)
 <>  BMWIDTH(bm)

      These functions return the depth, height, and width values for
      the bitmap.  These values are useful when doing blits.

      Inputs:

         bm - must be a pointer to a bitmap structure

      Returns: - depth, height, or width
```

1.16 IMG

```
 <>  IMGDEPTH(image)
 <>  IMGHEIGHT(image)
 <>  IMGWIDTH(image)

      These functions return the depth, height, and width values for
      the image.  These values are useful when doing blits.

      Inputs:

         image - must be a pointer to an image structure

      Returns: - depth, height, or width
```

1.17 FREEAREA

```
<>   FREEAREA(window)

      This function releases the memory allocated for the areafill/flood
      functions.

      Inputs:

          window  - pointer to a window opened with OPENWINDOW().
                    A prior call to MAKEAREA() should have been made
                    to the same window.

      Returns:    - always returns 1
```

1.18 FREE

```
<>   FREEBITMAP(pointertobitmap)
<>   FREEIMAGE(pointertoimage)
<>   FREERASTPORT(pointertorastport)
<>   FREEBIRASIM(pointer)
<>   FREETHISMENU(menustrippointer) (was FREEMENU)
<>   FREEITEXT(intuitextpointer)
<>   FREETHIS(pointer to any independent structure)

      These functions return the memory allocated for the bitmap, image
      rasterport, menustrip, intuitext, or whatever to the system.

      The APIG library maintains separate list for bitmaps, images,
      rastports, bitplane, and menustrip memory allocations.
      Using FREEBITMAP, FREEIMAGE, FREERASTPORT, FREETHISMENU, FREEITEXT
      will cause only the appropriate list to be searched, hopefully to
      save search time.

      FREEBIRASIM, will check only the bitmap, rastport, and image lists,
      until it finds the allocated structure or fails to find it.

      FREETHIS will check ALL list.

      Note independent requesters, gadgets, and borders are freed with
      the FREETHIS() function (after awhile this does get a bit redundant).


      Inputs:

         pointer - must be a pointer to an independent bitmap, image,
                   rastport or whatever.  Pointer must be obtained from
                   the corresponding make function (MAKEBITMAP(), etc.)

      Returns:   - returns positive non-zero value if successful, the
                   value will vary and is the number of 'things' that
                   were 'owned' by the object freed.

                   eg. FREEBITMAP() typically may return a value of 12,
                   the 12 represents the bitplane allocations and any
                   other allocations (eg. CMAP chunks) that were owned
                   by the bitmap.

                 - Returns zero if nothing was freed, ie. did not find
                   the structure.
```

1.19 FREE_EXEC_LIST

```
 <>  FREE_EXEC_LIST( listptr, nodestructsize, liststructsize )

      Traverses the list pointed to by listptr and frees the memory
      allocated to LN_NAME, the node structure, and the list structure.

      Inputs:

         listptr  - pointer, to Exec List structure

         nodesize - numeric, the size of each node on the list
                    This parameter is optional, if not specified
                    or zero it defaults to 14. If non-zero it
                    should be the same as the nodesize specifed
                    in ADD_LIST_NODE.

         listsize - numeric, the size of the list structure,
                    ie. the amount of memory that was allocated
                    to the list structure.

                    if 'listsize' = 0, then only the nodes are freed.
                    (the list structure 'listptr' is not freed)

                    ****
                    **** IF 'LISTSIZE' < 14, THEN SIZE IS SET TO 14 AND
                    **** USED IN FREEING THE LIST STRUCTURE.
                    ****

                    FREE_EXEC_LIST(list,0,1) - free nodes and list struct

                    FREE_EXEC_LIST(list,0,0) - free nodes Only

                    if 'listsize' > 13, then 'listsize' bytes are used
                    in freeing the list structure.

                    This parameter is optional, if not specified
                    default is zero, free nodes ONLY!!.

      Finally after freeing all nodes in the list, a NewList(listptr) is
      done if the list structure is not to be freed. Thus you can start
      re-building the list by adding new nodes.

      Returns:    - the number of nodes freed, this could be zero.

      See Also ADD_LIST_NODE.
```

1.20 FREE_EXEC_NODE

```
 <>  FREE_EXEC_NODE( nodeptr, nodesize )

      Frees the memory allocated to the node structure pointed to by
      'nodeptr'.  Note the node should have been previously Removed()
      from whatever list it is/was in.

      Inputs:

         nodeptr  - pointer, to Exec Node structure

         nodesize - numeric, the size of node structure.
                    This parameter is optional, if not specified
                    or zero it defaults to 14. If non-zero it
                    should be the same as the nodesize specifed
                    in ADD_LIST_NODE.

      Returns:    - returns sum total of bytes freed.

      See Also ADD_LIST_NODE.
```

1.21 GADSELECTED

```
 <>  GADSELECTED( gadgetptr )

      This function returns TRUE/FALSE depending on the state of
      the gadget.

      Inputs:

          gadgetptr  - pointer to a gadget

     Returns:        - returns 1 if the gadget state is SELECTED,
                       otherwise returns 0.
```

1.22 GETARRAY

```
 <>  GETARRAY(arrayptr,arrayindx)

      This function allows you to retrieve the 16-bit value stored in the
      array pointed to by 'arrayptr'.

      Inputs:

          arrayptr   - pointer to array (block of memory) as returned
                       by ARexx ALLOCMEM().

          arrayindx  - numeric, index position you wish to get

     Returns:        - returns value stored in array position 'arrayindx'.
```

```
    Also See: SETX(), SETY(), GETX(), GETY(), SETARRAY()
```

1.23 GETGADPTR

```
 <>  GETGADPTR( window,gadgetid,requester )

      This function searches the window's gadget list for a gadget with
      GadgetID equal to 'gadgetid' and returns a pointer to the gadget.

      Inputs:

          window     - pointer to window opened with OPENWINDOW().

          gadgetid   - numeric, gadgetid

          requester  - pointer to a requester, if a requester gadget
                       else code as zero

    Returns:         - pointer (ARexx hex-string) to gadget.
```

1.24 GETIDCMP

```
 <>  GETIDCMP(window)

      This function allows you to retrieve the current value of IDCMPFlags
      for the window.

      Inputs:

          window     - pointer to window as returned by OPENWINDOW().

    Returns:         - windows IDCMPFlags value
```

1.25 GETLAYERINFO

```
 <>  GETLAYERINFO(layer)

      This function allows you to retrieve the pointer to the layers
      layer_info structure.

      Inputs:

          layer      - pointer to layer

    Returns:         - pointer to layer info for the layer
```

1.26 GETLAYERRASTPORT

```
 <>  GETLAYERRASTPORT(layer)

      This function allows you to retrieve the pointer to the layers
      RastPort. You will need the rastport pointer for the layer when
      using the drawing functions.

      Inputs:

          layer      - pointer to layer as returned by CREATEUPFRONTLAYER()
                       or CREATEBEHINDLAYER() functions.

    Returns:         - pointer to rastport for the layer
```

1.27 GETRPBITMAP

```
  <>  GETRPBITMAP(rp)

      This function allows you to retrieve the pointer to the BitMap of
      the RastPort.

      Inputs:

          rp         - pointer to RastPort structure.

    Returns:         - pointer to bitmap for the rastport
```

1.28 GETSCREENBITMAP

```
 <>  GETSCREENBITMAP(screen)

      This function allows you to retrieve the pointer to the BitMap of
      the screen.

      Inputs:

          screen     - pointer to screen, returned by OPENSCREEN().

    Returns:         - pointer to bitmap for the rastport
```

1.29 ETSCREENRASTPORT

```
 <>  GETSCREENRASTPORT(screen)

      This function allows you to retrieve the pointer to the RastPort of
      the screen.

      Inputs:
          screen     - pointer to screen, returned by OPENSCREEN().

     Returns:        - pointer to rastport for the screen
```

1.30 GETSTRGAD

```
 <>  GETSTRGAD( window,gadgetid,requester )

      This function retrieves the value of the string gadget.

      Inputs:

          window     - pointer to window opened with OPENWINDOW().

          gadgetid   - numeric, gadgetid of the string gadget

          requester  - pointer to a requester, if a requester gadget
                       else code as zero

     Returns:        - string contents of the string gadget
```

1.31 GETVALUE

```
 <>  GETVALUE(ptr,offset,size,type)

      This function allows you to retreive the value of any parameter in
      any data structure.

      ptr    - pointer, (ARexx hex string) to any data structure, ie. window
               screen, bitmap, etc.

      offset - numeric, specifies the relative position, from the beginning
               of the data structure, of data value you want to retrieve.
               (see RKM or include '.i' files for offsets)

      size   - numeric, specifies the size of the data value you want to
               retrieve.  This value must be either 1, 2, or 4.  Any other
               value will cause the function to return a NULL ('0000 0000'x).

      type   - string, either a 'N', 'P' or 'S', this specifies the type
               of data you are retrieving.

               'N' specifies that you want the value returned as a numeric.
                   for sizes of 1 and 2 the returned value is always as a
                   numeric.

               If the size is 4 then you can also use (in addition to 'N')
               the following:

               'P' specifies that you want the value returned as a pointer.
                   (ie. ARexx hex string)
```

```
'S' specifies that you want the value returned as a string.

    VERY IMPORTANT NOTE !!!
    When using 'P' or 'S',  'ptr' + 'offset' must result
    in an address which contains a pointer to something.


eg.  You can retrieve the window title string with:

     title = getvalue(windowpointer,32,4,'S')

     say "Your Window title is" title

     (the title string pointer is offset 32 from the beginning
      of the window structure)


eg.  To have the window title pointer returned as a pointer:

     titleptr = getvalue(win,32,4,'p')

     say "Window Title Pointer is " d2x(c2d(titleptr))


eg.  To return the contents of a string gadget :

     specialstringinfo = getvalue(gadgetpointer,34,4,'P')
     gadcontents = getvalue(specialstringinfo,0,4,'S')
     say "Your gad string is" gadcontents

eg.  Determining if a gadget is SELECTED:

     if bittst(d2c(getvalue(gadpointer,12,2,'N')),7) = 1 then
        say "Gadget SELECTED"
     else
        say "Gadget NOT SELECTED"

     getvalue(gadpointer,12,2,'N') returns the gadget Flags
     d2c() converts the result to form 'nnnn'x
     bittst() test bit 7, the gad select bit.



returns -  the value you specified in the form specified.
           defaults to returning numeric if type not 'N', 'P', or 'S'
           returns NULL ('0000 0000'x) if size not 1, 2, or 4.
           (their is no way to distinguish between returning a valid
            null pointer ('P' type) and error in size/type)
```

1.32 GETWINDOWLAYER

```
<>   GETWINDOWLAYER(window)

     This function allows you to retrieve the windows layer pointer.
```

```
      Inputs:

          window     - pointer to window as returned by OPENWINDOW().

     Returns:        - pointer to rastport for the layer
```

1.33 GETWINDOWRASTPORT

```
 <>  GETWINDOWRASTPORT(window)

      This function allows you to retrieve the windows rastport pointer

      Inputs:

          window     - pointer to window as returned by OPENWINDOW().

     Returns:        - pointer to rastport for the window
```

1.34 GETX

```
 <>  GETX(arrayptr,xindex)

      This function does the same as GETARRAY(), the difference is that
      it computes the index offset value for the X-pair for you.

      eg. if arrayptr is an array of 20 XY-pairs (40 16-bit values)
          then
               x = getx(arrayptr,8), retrieves the value for the 8th
                   X value.
               the equivalent using GETARRAY() would be
               x = getarray(arrayptr,32)

      Inputs:

          arrayptr   - pointer to array as returned by ARexx ALLOCMEM().

          xindex     - numeric, index position you wish to set

     Returns:        - returns the value stored in array position xindex.

     Also See: SETX(), SETY(), GETY(), GETARRAY(), SETARRAY()
```

1.35 GETY

```
 <>  GETY(arrayptr,yindex)

      This function does the same as GETARRAY(), the difference is that
      it computes the index offset value for the Y-pair for you.
      Inputs:

          arrayptr   - pointer to array as returned by ARexx ALLOCMEM().

          yindex     - numeric, index position you wish to retreive.

     Returns:        - returns the value stored in array position yindex.

     Also See: SETX(), SETY(), GETY(), GETARRAY(), SETARRAY()
```

1.36 HORIZ

```
 <>  HORIZPOT(propgadgetptr) <>  HORIZBODY(propgadgetptr)

      These functions return the value of the horizontal components of the
      pot gadget.

      Inputs:

      propgadgetptr  - pointer to a proportional gadget, return from
                       MAKEPROPGADGET().

     Returns:        - returns the numeric value of HorizPot
```

1.37 IFF

```
 <>  IFFDEPTH(pointer) <>  IFFHEIGHT(pointer) <>  IFFWIDTH(pointer)

      These functions return the various dimensions of the IFF image
      pointed to by the bitmap pointer.

      Note that BMDEPTH()/BMHEIGHT/BMWIDTH() will return the same values.
      The difference is where the information is obtained, the BM...
      functions retrieve the values from the bitmap itself.  The IFF...
      functions retrieve the values from the 'BMHD' chunk that was loaded
      with the IFF file.  The two sets of values should always be the
      same since the 'BHMD' chunk values are used to allocate the bitmap.
      (Unless, of course, the IFF was blitted into a different bitmap)


      Inputs:

          pointer    - pointer to a IFF bitmap (returned by LOADIFF()).
                       This must be the original bitmap pointer
                       (or copy of it) returned by the LOADIFF() function.
                       You cannot use a pointer returned by MAKEBITMAP()
                       into which you have blitted the IFF.  If the pointer
                       was returned from MAKEBITMAP() then use BMDEPTH() etc.

      Returns:       - returns the width, height, depth of the IFF
                       image.
```

1.38 IFFVIEWMODE

```
<>  IFFVIEWMODE(pointer)

      This function returns the viewmodes (HAM/HIRES/LACE etc.) word
      for the IFF.

      Inputs:

          pointer    - pointer to a IFF bitmap (returned by LOADIFF()).
                       This must be the original bitmap pointer
                       (or copy of it) returned by the LOADIFF() function.
                       You cannot use a pointer returned by MAKEBITMAP()
                       into which you have blitted the IFF.

      Returns:       - returns the viewmodes word
```

1.39 IFFCOLORS

```
<>  IFFCOLORS(pointer)

       This function returns the number of colors used in the IFF

      Inputs:

          pointer    - pointer to a IFF bitmap (returned by LOADIFF()).
                       This must be the original bitmap pointer
                       (or copy of it) returned by the LOADIFF() function.
                       You cannot use a pointer returned by MAKEBITMAP()
                       into which you have blitted the IFF.

      Returns:       - returns number of colors
```

1.40 IFFCOLORTAB

```
<>  IFFCOLORTAB(pointer)

       This function returns a pointer to the color table used by the IFF.

      Inputs:

          pointer    - pointer to a IFF bitmap (returned by LOADIFF()).
                       This must be the original bitmap pointer
                       (or copy of it) returned by the LOADIFF() function.
                       You cannot use a pointer returned by MAKEBITMAP()
                       into which you have blitted the IFF.

      Returns:       - returns a pointer (ARexx string) to a color table
                       (array of short integers) which specify the color
                       values used in the IFF.
                       returns 0 if not a pointer to IFF or if the IFF
                       does not contain a CMAP chunk.
```

1.41 LOADIFF

```
<>  LOADIFF(filename,owner)

     This function uses Christian Weber's iff.library to load an IFF file.

     Inputs:

         filename   - the name of the IFF file.


         owner      - pointer to object which will own this IFF.

                      if 0 then this will be an independent
                      IFF bitmap which must be explictly freed.

                      Owner can also be ANY of the owners described
                      in the MAKEBOOLGADGET description.

     Returns:       - This function returns a pointer to the bitmap
                      structure containing the IFF imagery.  Note that
                      no display of the IFF imagery is done.  To display
                      the imagery use one of the blit functions contained
                      in this library, to blit it into the bitmap/rastport
                      of your choice. When you are done using the bitmap
                      imagery you should free the memory allocated to it,
                      by calling FREEBITMAP() or FREEBIRASIM().

                    - returns null ('0000 0000'x) if load fails
```

1.42 LOADIMAGE

```
<>  LOADIMAGE(filename,imageptr,left,top,owner)

     This function uses Christian Weber's iff.library to load an IFF file.
     The difference between this function and LOADIFF() is that this
     function returns a pointer to an Image structure.  The bitmap planes
     of the IFF file are re-arranged to conform with the layout required
     for ImageData.

     Inputs:

         filename   - the name of the IFF file.

         imageptr   - pointer to image structure
                      if this is non-null the loaded image
                      will be linked to this image.

         left       - numeric, the value to set the LeftEdge of
                      the image structure

         top        - numeric, the value to set the TopEdge of
                      the image structure
```

```
    owner      - pointer to object which will own this Image.

                 if 0 then this will be an independent
                 image which must be explictly freed.

                 Owner can also be ANY of the owners described
                 in the MAKEBOOLGADGET description.

Returns:       - This function returns a pointer to an Image structure
                 containing the IFF imagery.  Note that no display of
                 IFF imagery is done.  To display the imagery you use
                 the DRAWIMAGE() function.  The returned value can
                 also be used in defining imagery for gadgets.
                 When you are done using the Image you should free
                 the memory allocated to it, by calling FREEIMAGE()
                 or FREEBIRASIM().

               - returns null ('0000 0000'x) if load fails
```

1.43 MAKEAREA

```
 <>  MAKEAREA(window,xsize,ysize,maxvectors)

      Initializes the windows rastport for use with the Area-Fill-Flood
      functions.  This function consumes a large chuck of your CHIP memory
      you should get your area fill operations done as quickly as possible
      then free up the memory with the function FREEAREA().
      This function must be called before you do any areafill/floodfills.
      The memory remains allocated until FREEAREA() is called.


      Inputs:

          window     - pointer to window opened with OPENWINDOW().

          xsize      - numeric, specifies the width of the largest area you
                         plan to fill.

          ysize      - numeric, specifies the height of the larges area you
                         plan to fill.

                         (Recommended that xsize,ysize be the same size
                          as the rasport you are drawing into.)

          maxvectors - numeric, specifies the number of vertices (points)
                         you expect to draw.  This parameter is NOT ignored
                         and must be non-zero.

    Returns:         - always returns 1, this simply indicates that the
                         library made the call to the Intuition/Graphics
                         library function.
```

1.44 MAKEBITMAP

```
<>  MAKEBITMAP(width,height,depth,owner)

     This function allocates and builds a bitmap structure and returns
     a pointer to it.  The purpose for this is to allow you to build
     off-screen bitmaps then blit them on screen when ready.

     Inputs:

         width       - width of the bitmap

         height      - height of the bitmap

         depth       - depth, number of bit planes, of the bitmap

         owner       - pointer to object which will own this bitmap.
                       code as zero if independent bitmap, in which
                       case it must be explicitly freed, with FREEBITMAP
                       or FREEBIRASIM.

                     - pointer  Any other independent owner see
                       description of owners under MAKEBOOLGAGET.
                       Their are no restrictions on who/what can
                       own a bitmap.

     Returns:        - pointer to bitmap structure
                     - returns null ('0000 0000'x) if call fails
```

1.45 MAKEBOOLGADGET

```
<>  MAKEBOOLGADGET( owner,left,top,width,hgt,flags,activation,
                              itext,bpen,render,select,gadid,linkto )

     Allocates and builds a boolean gadget structure.

     Inputs:

         owner       - pointer to object which will own the memory
                       allocated for this gadgets structure.

                       For gadgets the owner can be any one of the
                       following:

                       0).  0 - independent structure
                            If the 'owner' is coded as zero then the
                            gadget created can be used as an owner
                            for other structures.

                       1).  pointer to window
                            The gadget will be owned by this window,
                            the gadget memory will be freed when
                            the window is closed. NOTE THE GADGET IS
                            NOT PLACED IN THE WINDOWS GADGET LIST.
```

```
                        2).  pointer to screen
                             The gadget will be owned by this screen,
                             the gadget memory will be freed when
                             the screen is closed.

                        3).  pointer to requester
                             The gadget will be owned by this requester,
                             the gadget memory will be freed when the
                             requester is freed. NOTE THE GADGET IS
                             NOT PLACED IN THE REQUESTERS GADGET LIST.

                        4).  pointer to intuitext
                             The gadget will be owned by this intuitext,
                             the gadget memory will be freed when the
                             intuitext is freed.

                        5).  pointer to gadget
                             the created gadget will be owned by the
                             specified gadget, the created gadget memory
                             will be freed when the specified owner
                             gadget is freed. NOTE THE GADGETS ARE NOT
                             LINKED.

                        6).  pointer to border
                             The gadget will be owned by this border,
                             the gadget memory will be freed when the
                             border is freed.

                        7).  pointer to menustrip
                             The gadget will be owned by this menustrip,
                             the gadget memory will be freed when the
                             menu is freed.


          left       - numeric, left edge placement of gadget

          top        - numeric, top edge placement of gadget

          width      - numeric, width of gadget in pixels

          hgt        - numeric, height of gadget in pixels

          flags      - numeric, gadget flags
                        GADGHCOMP/GADGIMAGE/GRELRIGHT/GRELWIDTH/SELECTED/etc.

          activation - numeric, IDCMP flags for the gadget
                        TOGGLESELECT/GADGIMMEDIATE/RELVERIFY/etc.

          itext      - pointer to intuitext to be displayed with the gadget.
                        Note that this can be a linked list of IntuiText.
                        Keep in mind that the position of the IntuiText is
                        relative to the gadgets left & top coordinates.

          bpen       - numeric, border pen color
                        this function will build a border (only if 'render'
                        parm is zero) to 'box' the gadget, this is the color
```

```
                    you want the border lines to be.

      render      - pointer to Border/Gadget imagery.
                    If this is zero then APIG will build a default
                    border structure to 'box' the gadget.
                    If this is non-zero then APIG assumes this is a
                    pointer to an appropriate structure for the
                    GadgetRender parm and uses it as-is.  You may still
                    link a Border/Image structure in afterwards.
                    If this is -1 then APIG will leave the GadgetRender
                    parm of the Gadget structure NULL, you can then link
                    a Border/Image structure in afterwards.

      select      - pointer to Border/Gadget imagery to display when
                    selected.  APIG uses what ever you specify so
                    do not put a -1 here, put a 0 if you want it
                    to be NULL.

      gadid       - numeric, any value you wish to identify this gadget.

      linkto      - pointer to GADGET to which this gadget should be
                    linked this allows you to string several gadgets
                    together into a linked list.  The new allocated
                    gadget will be placed at the end of the list.

                    You may also specify a pointer to a WINDOW or a
                    pointer to a REQUESTER as the 'linkto'.

                    If 'linkto' is a window pointer then APIG will
                    AddGadget() the new gadget to the windows gadget
                    list. You however must do gadget refreshing.

                    If 'linkto' is a requester pointer then APIG will
                    place the new gadget at the end of the requesters
                    gadget list. Be sure to specify appropriate flags
                    for requester gadgets, eg. ENDGADGET.

                    (APIG will set the gadget type to REQGADGET if the
                     gadget is placed in a requester)

 Returns:         - pointer to allocated gadget as an ARexx hex string.
                  - returns null ('0000 0000'x) if call fails
```

1.46 MAKEBORDER

```
 <>  MAKEBORDER( owner,arrayptr,arraycnt,left,top,fp,bp,dm,linkto)

      Allocates and builds a border structure.

      Inputs:

          owner       - pointer to object which will own this border.

                        if 0 then this will be an independent
                        border which must be explictly freed.
```

```
                 Owner can also be ANY of the owners described
                 in the MAKEBOOLGADGET description.

    arrayptr   - pointer to array of short integers,
                 taken as XY pairs.
                 The array can be allocated with ALLOCMEM(),
                 and the XY pair values set with SETX() and
                 SETY() functions.

                 If you code a zero here then MAKEBORDER()
                 will generate a border array for you.
                 See parms below.

    arraycnt   - numeric, number of XY pairs in the array,
                 must be less than 128.

                 If 'arrayptr' was specified as zero then
                 this parm will be the 'thickness' of the
                 border. If this is the case then 'arraycnt'
                 must be <= 25. A value of 0 defaults to 1.

    left       - numeric, borders left edge

                 If 'arrayptr' was specified as zero then
                 this parm will be the width of the border.

    top        - numeric, borders top edge

                 If 'arrayptr' was specified as zero then
                 this parm will be the heigth of the border.

    fp         - numeric, borders front pen color

    bp         - numeric, borders back pen color

    dm         - numeric, drawmode to use

    linkto     - pointer to a BORDER structure to which the new
                 border will be linked. Code this as zero if
                 not linking the border.

                 You may also specify a pointer to a GADGET or a
                 pointer to a REQUESTER as the 'linkto'.

                 If 'linkto' is a GADGET pointer then APIG will
                 place the new border at the end of the gadgets
                 GadgetRender list.  This implies that gadget has
                 a GadgetRender list consisting only of borders
                 or the gadgets GadgetRender pointer is NULL.
                 Note that the SelectRender pointer is not
                 modified.

                 If 'linkto' is a REQUESTER pointer then APIG will
                 place the new border at the end of the requesters
                 ReqBorder border list.
```

```
Returns:        - pointer to allocated Border structure.
                - returns null ('0000 0000'x) if call fails
```

1.47 MAKEITEM

```
 <>  MAKEITEM( menustrip,text,menu,left,top,width,height,flags,
                         ME,COM,fp,bp,dm,itemfill,selectfill )


      Builds and attaches a menu-item to a menu
      All arguments are the same as MAKESUBITEM.

      See MAKESUBITEM below, also see menu hints.
```

1.48 MAKEITEXT

```
 <>  MAKEITEXT( owner,text,xpos,ypos,fpen,bpen,dmode,fontattr,linkto )

      Allocate and build an IntuiText structure.

      Inputs:

          owner      - pointer to object which will own this IntuiText.

                       if 0 then this will be an independent
                       IntuiText which must be explictly freed.

                       Owner can also be ANY of the owners described
                       in the MAKEBOOLGADGET description.


          text       - string, text you want to build IntuiText for

          xpos       - numeric, x displacement in pixels

          ypos       - numeric, y displacement in pixels

          fpen       - numeric, foreground pen color

          bpen       - numeric, background pen color

          dmode      - numeric, draw mode (ie. JAM1,JAM2, etc.)

          fontattr   - pointer, to text font attributes
                       (returned by MAKETATTR)
                       code as zero to get default

          linkto     - pointer, to IntuiText structure to which this
                       IntuiText should be linked.

                       You may also specify a pointer to a GADGET,
                       pointer to a REQUESTER, pointer to a MENUITEM,
                       or a pointer to a MENUSUBITEM as the 'linkto'.

                       If 'linkto' is a GADGET pointer then APIG will
                       place the new IntuiText at the end of the gadgets
                       GadgetText list.

                       If 'linkto' is a REQUESTER pointer then APIG
                       will place the new IntuiText at the end of the
                       requesters ReqText list.

                       If 'linkto' is a MENUITEM pointer then APIG
                       will place the new IntuiText at the end of the
                       MENUITEMs ItemFill list.  This implies that the
                       ItemFill of the MENUITEM consist of IntuiText
                       and is not an IMAGE. MENUSUBITEMS are treated
                       the same way.  Note the SelectFill is not
                       modified.

     Returns:        - pointer to the allocated IntuiText
                     - returns null ('0000 0000'x) if call fails
```

1.49 MAKEMENU

```
 <>  MAKEMENU( menuowner,menutext,leftedge,width,flags,menupointer )

      This function creates/builds a menu header.

      The parm 'menupointer' determines whether a new menustrip is
      being constructed or whether a new header is being added to
      an existing menustrip.

      The initial call to MAKEMENU should set the parm 'menupointer'
      to 0, this indicates the start of a new menustrip.  The initial
      call builds a MENU structure and returns a pointer to it.

      All subsequent calls MUST then use the return value from the
      initial call to add new menu headers to the menustrip.  Also
      remember that items/subitems will need an owner, it is this
      initial value that you will use. (if not using a window/screen)

      eg. /* the last parm is 0, indicating start a new menustrip */
          /* the first parm is 0, indicating independent menu     */
          menu0 = MAKEMENU( 0,"Project",left,width,flags,0 )
          menustrip = menu0

      eg. /* the last parm is 0, indicating start a new menustrip */
          /* the first parm is window, indicating menu is owned   */
          /* by the window, and will be freed when window closes  */
          menu0 = MAKEMENU( window,"Project",left,width,flags,0 )
          menustrip = menu0

          to add additional menu headers to the menustrip the
          return value menu0 must now must be used as the
          'menupointer' parm.
```

```
eg. /* the last parm is non-null, indicating link the */
    /* new menu to it, owner simply says who owns the */
    /* new allocated menu structure memory            */
    menu1 = MAKEMENU( menu0,"This",left,width,flags,menu0 )
    menu2 = MAKEMENU( menu0,"That",left,width,flags,menu0 )
    or
    menu3 = MAKEMENU( menustrip,"Whatever",left,width,flags,menu0 )
     (since menustrip was assigned the value of menu0)

    menu headers which are not owners returned from the initial
    call cannot be used to build the menustrip.

eg. menu4 = MAKEMENU( menu1,"Lastone",left,width,flags,menu0 )

    the above will NOT work, menu1, was not returned from the initial
    call, result is a null pointer for the value of menu4.


This differs from APIG version 0.5, and allows you to re-use
menu strips after the window has been closed.

You must now explicitly release memory that has been allocated
to independent menustrips, using the FREETHISMENU() function.


Inputs:

    menuowner     - pointer to object which will own the MENU.

                  For menus valid owners can only be:

                  0).  0 - code as zero when creating new independent
                       menustrip.  You will need to free it with
                       FREETHISMENU.

                  1). pointer to window

                  2). pointer to screen

                  3). pointer to independent MENU

                  Menus cannot be owned by any other structure.
                  ie.  gadgets, intuitext, etc. cannot be owners
                  of a menu.


    menutext    - string, text to be displayed in menu header

    leftedge    - numeric, location of menu select box.
                  (measured in pixels)

                  If leftedge >= 0 then the left edge is
                  measured from the right edge of the
                  MENU which PRECEDES it.

                  If leftedge < 0 then the left edge is
                  measured from the leftedge of the screen.
```

```
           +--------+ +--------+     +----------+
           | MENU 1 | | MENU 2 | ... | MENU N-1 |
           +--------+ +--------+     +----------+
           |                                    |
           |                                    |
           |                                    |<--- meas.
left       |                                    |from here if
edge       |<--- meas.                          |’leftedge’ >= 0
screen --> |from here                           |for MENU N
           | if ’leftedge’ < 0                  |
           | for MENU N                         |
           |                                    |
```

```
    width       - numeric specifies width of MENU select box.
                  (measured in pixels)
                  If zero, then the IntuiTextLength of the
                  ’menutext’ is used.
                  If ’menutext’ is a NULL string then the
                  IntuiTextLength of 1 character is used.


    flags       - numeric, this is the Flags variable of the
                  Menu structure, only meaningful values are
                  MENUENABLED or 0 (menu will be disabled).


    menupointer - pointer to a menustrip in which this menu
                  should be linked.  If this is zero then
                  it indicates you are building a new menu
                  strip.

    Returns:    - pointer to menu.
                - null (’0000 0000’x) if call fails


Note that the menu strip is not attached to the window, the menu
is simply constructed.  You will need to call SetMenuStrip to
actually attach the menu strip to the window.
```

1.50 MAKENEWGADGET

```
<>  MAKENEWGADGET(vinfo,font,left,top,width,height,text,flags,id,usrdata)

    Allocate and build a NewGadget structure for use with GadTool
    gadgets.  Note this cannot be owned, must be explicitly freed
    with the FREETHIS() function.

    Inputs:

    vinfo     - pointer to visual info, obtained from prior call
```

```
                 to GETVISUALINFO() function.
                 (ie. vinfo = GETVISUALINFO(screen)
                 a value for vinfo is REQUIRED!

     font      - pointer to font to be used for gadget.
                 can be obtained from screen with:
                 font = GETVALUE(screen,40,4,'p')
                 a value for font is REQUIRED!

     left      - numeric, left edge placement of gadget

     top       - numeric, top edge placement of gadget

     width     - numeric, width of gadget

     height    - numeric, height of gadget

     text      - string, text to be used for gadget label

     flags     - numeric specifies placement of gadget label text
                 (PLACETEXT_LEFT/PLACETEXT_IN/PLACETEXT_ABOVE/etc.)

     id        - numeric, value to be used for gadget ID

     usrdata   - pointer, to whatever you want to point to.


     Returns   - pointer to allocated NewGadget structure


     NOTE: A buffer (ng_GadgetText) is allocated to hold the 'text' string
           supplied, this buffer is 'owned' by the NewGadget structure, and
           will be freed when the NewGadget structure is freed.  Therefore
           do not free the NewGadget structure until you free the gadtool
           gadgets. If you free the NewGadget structure before freeing the
           gadgets you wil also free the ng_GadgetText pointers used by
           those gadgets.

     See Also SETNEWGADGET()
```

1.51 MAKENEWMENU

```
 <>  MAKENEWMENU(n)

    This function allocates an internal structure used by APIG to
    allow you to dynamically build NewMenu structures for use with
    the GADTools CREATEMENU() functions.

      Inputs:

          n          - numeric, initial number of NewMenu array entries
                         to allocate.  This parameter is optional if not
                         specified or zero, then the default value is 20.

                         eg.
```

```
                    apignmdata = MAKENEWMENU()

                    would be like

                    struct NewMenu mymenu[20];    in 'C'

      Returns     - pointer, to internal structure, which, for your
                    informational purposes only, looks like:

                    STRUCTURE APIGNM,0
                              WORD  nmcount

                              WORD  nminuse

                              LONG  nmsize

                              APTR  nmpointer  ; actual NewMenu entries
                                               ; this value will change
                                               ; when re-allocated

                              LONG  nmspare1   ; some day ...

                              LABEL apignm_SIZEOF
                    If you modify any values in this structure you
                    deserve to crash and burn.
```

You use the ADDTO_NEWMENU() function to populate the array entries.
You do not need to know exactly how many NewMenu entries you will
need. ADDTO_NEWMENU() will allocate more entries if you exceed
the initial allocation, ADDTO_NEWMENU() will re-allocate as many
times as neccessary.
Specifying a 'n' value close to the number of entries you expect,
reduces the number of times the array has to be re-allocated and
re-built.

To free the memory allocated you use the FREETHIS() function.
However you cannot free the memory until you are completely done
using the menu. The 'apignmdata' is seen as an independent owner
and owns everything that was allocated by ADDTO_NEWMENU().

```
eg.
    apignmdata = MAKENEWMENU(100)
    call ADDTO_NEWMENU(apignmdata,...)
              .
              .
              .
    call ADDTO_NEWMENU(apignmdata,...)
    realmenu = CREATEMENU(apignmdata,...)
    call LAYOUTMENU(realmenu,...)
    mywindow = OPENWINDOW(...)
    call SETMENUSTRIP(mywindow,realmenu)
    call CLOSEWINDOW(mywindow)
    call FREEMENUS(realmenu)
    call FREETHIS(apignmdata)
```

```
/* or FREETHISMENU(apignmdata), because its on the MENULIST */
```

1.52 MAKEPROPGADGET

```
<>  MAKEPROPGADGET( owner,left,top,width,hgt,flags,activation,itext,
                     ,piflags,hbody,vbody,gadid,linkto,knobimage)

     Allocate and build a proportional gadget structure.
     (Note parms are different than version 0.5, hbody/vbody/knobimage)

     Inputs:

         owner      - pointer to object which will own this Gadget.
                      See description of MAKEBOOLGAGET.

         left       - numeric, left edge placement of gadget

         top        - numeric, top edge placement of gadget

         width      - numeric, width of gadget in pixels

         hgt        - numeric, height of gadget in pixels

         flags      - numeric, gadget flags

         activation - numeric, IDCMP flags for the gadget

         itext      - pointer to intuitext to be displayed with the gadget

         piflags    - numeric, specifies the prop gadgets characteristics
                      AUTOKNOB/FREEHORIZ/FREEVERT/KNOBHIT/PROPBORDERLESS

         hbody      - numeric, specifies the horizontal step (percentage)
                      amount

         vbody      - numeric, specifies the vertical step (percentage)
                      amount

         gadid      - numeric, any value you wish to identify this gadget

         linkto     - pointer, gadget to which this gadget should be
                      linked.
                      See description of MAKEBOOLGADGET all the same
                      applies.

         knobimage  - pointer, this points to an image. This image will be
                      used as the prop gadgets knob.  This allows you to
                      use custom slider knobs in your proportional gadgets
                      instead of just a rectangluar knob.

    Returns:        - pointer to allocated gadget as a rexx hex string.
                    - returns null ('0000 0000'x) if call fails

         Note: HORIZPOT & VERTPOT will be initialized to zero.
```

1.53 MAKERASTPORT

```
 <>  MAKERASTPORT(width,height,depth,owner)

       This function allocates and builds a rastport structure and returns
       a pointer to it.  The purpose for this is to allow you to build
       off-screen rastports then blit them on screen when ready.

       Inputs:

           width       - numeric, width of the rastports bitmap

           height      - numeric, height of the rastports bitmap

           depth       - numeric, number of bit planes in rastports bitmap

           owner       - pointer to object which will own this rastport.
                         code as zero if independent rastport, in which
                         case it must be explicitly freed, with FREERASTPORT
                         or FREEBIRASIM.

                       - pointer  Any other independent owner see
                         description of owners under MAKEBOOLGAGET.
                         Their are no restrictions on who/what can
                         own a rastport.

       Returns:        - pointer to rastport structure
                       - returns null ('0000 0000'x) if call fails
```

1.54 MAKEREQUESTER

```
 <>  MAKEREQUESTER(window,left,top,width,height,gadget,text,border,
                              backfill,flags,relleft,reltop,bm)

       This function builds an instance of a requester structure to be
       used with the REQUEST() function.

       Inputs:

           owner       - pointer to object which will own this Requester.
                         See description of MAKEBOOLGAGET.

           left        - numeric, left edge offset of the requester

           top         - numeric, top edge offset of the requester

           width       - numeric, width of the requester

           height      - numeric, height of the requester

           gadget      - pointer to first gadget of list of gadgets within
                         the requester.  YOU should also make sure that
                         at least one of the gadgets in the list has its
                         ENDGADGET flag set.
```

```
                  APIG will set the gadgettype flag of all gadgets in
                  this list to REQGADGET.

        text       - pointer to intuitext for the requester

        border     - pointer to a border structure

        backfill   - numeric, pen color to fill requester, before graphics
                  are rendered

        flags      - numeric, requester flags specify POINTREL,PREDRAWN
                  otherwise code as zero

        relleft    - numeric, left offset for requesters displayed
                  relative to the mouse pointer (ie. POINTREL set)

        reltop     - numeric, top offset for requesters displayed
                  relative to the mouse pointer (ie. POINTREL set)

        bm         - pointer to a bitmap containing imagery for the
                  requester, the PREDRAWN flag must be set.
                  Code as zero if not using your own imagery.

    Returns:       - pointer to a requester structure
                   - null ('0000 0000'x) if call fails
```

1.55 MAKESTRGADGET

```
 <>  MAKESTRGADGET( window,left,top,width,hgt,flags,activation,
                     itext,bpen,render,select,gadid,linkto,strlen,undobu f)

    Allocate and build a string gadget structure.

    Inputs:

        owner      - pointer to object which will own this Gadget.
                  See description of MAKEBOOLGAGET.

        left       - numeric, left edge placement of gadget

        top        - numeric, top edge placement of gadget

        width      - numeric, width of gadget in pixels

        hgt        - numeric, height of gadget in pixels

        flags      - numeric, gadget flags

        activation - numeric, IDCMP flags for the gadget

        itext      - pointer to intuitext to be displayed with the gadget

        bpen       - border pen color
                  this function will build a border structure
                  (only if 'render' parm is NULL/ZERO) to 'box'
```

```
                       the gadget, this is the color you want the border
                       lines to be.

        render       - pointer to Border/Gadget imagery.
                       If this is zero then APIG will build a default
                       border structure to 'box' the gadget.
                       If this is -1 then APIG will leave the GadgetRender
                       parm of the Gadget structure NULL, you can then link
                       a Border/Image structure in afterwards.

        select       - pointer to Border/Gadget imagery to display when
                       selected.  APIG uses whatever you specify so
                       do not put a -1 here, code a 0 if you want it
                       to be NULL.

        gadid        - numeric, any value you wish to identify this gadget

        linkto       - pointer, gadget to which this gadget should be
                       linked.
                       See description of MAKEBOOLGADGET all the same
                       applies.

        strlen       - numeric, max number of characters allowed in gadget

        undobuf      - pointer, points to memory block allocated for use as
                       the string gadgets UndoBuffer.  The memory may be
                       allocated using either ARexx's ALLOCMEM() or APIG's
                       MAKEPOINTER() function.  This parm is not validated.

    Returns:         - pointer to allocated gadget as a rexx hex string.
                     - returns null ('0000 0000'x) if call fails
```

1.56 MAKESUBITEM

```
 <>  MAKESUBITEM( menustrip,text,item,left,top,width,height,flags,
                          ME,COM,fp,bp,dm,itemfill,selectfill )

     This function builds and attaches a sub-item to a menu-item.
     This description also applies to MAKEITEM().


     Inputs:

        menustrip  - pointer to a menustrip, returned from initial call
                     of MAKEMENU().


        text       - text string, item text to be used
                     this parm along with fp,bp, and dm parameters will
                     be used to build an IntuiText structure.
                     The 'itemfill' variable of the MenuItem structure
                     will be initialized to point to the IntuiText
                     constructed.
```

```
item       - pointer to a menu-item, as returned by MAKEITEM()
             function. This is the menu-item you want to attach
             the sub-item to

             For MAKEITEM() this is menu pointer returned by
             MAKEMENU()


left       - numeric, left offset relative to menu select box

             The value you specify here is critical to the
             placement of the item select box, this value
             determines where the LEFTEDGE of the select box
             will be placed.
                 +--------------------+
                 |       MENU         |
                 |      HEADER        |
                 |        BOX         |
                 +--------------------+
                 |                    |
                 |                    |
                 |                    |
                 |from                |
                 |this                |
                 |edge                |
                 |                    |
            -x   |    +x              |
           <--- out --->              |
                 |                    |
                 |                    |
                 +----------+
                 | MENUITEM | menuitem will be positioned
                 +----------+ relative menu box.

                 +----------+
                 | SUB ITEM | subitem will be positioned
                 +----------+ relative menu box.



top        - numeric, top offset relative to menu select box
             the value you specify here is critical to the
             placement of the item select box.

             if 'top' < 65535 then the select box is measured
             from the bottom of the previous menu.

             eg.  top = 0, places top edge of select box
                  immediately below the item/subitem that
                  precedes it.

                  top = 20, places top edge of select box
                  20 pixels below the item/subitem that
                  precedes it.


             if 'top' < 0 then the select box is measured from
```

```
                   the top edge of the previous menu item.
                   (the height of the previous item is NOT included)


                   if 'top' > 65535 then the select box is measured
                   from the top edge of the menu (modulo 65536).

                     +----------------+
                             .
                             .

                         MENU ITEMS
                             .
                             .

                     +----------------+ top <= -1 measusre
       previous      | ^  MENU/ITEM   | from here down
       menu item     | |              |
                     | |              |
                     | height         |
                     | |              |
                     | |              |
                     | v              |
                     +----------------+ 0 <= top < 65535 measure
                                        from here down

                            +-----------+ <--- this edge will be
                            |  SUB ITEM |      relative to one of
                            +-----------+      the edges above,
                                               depending on value
                                               of 'top'

     width      - numeric, width of the item select box
                  code as zero to get default, the default width is
                  the IntuiTextLength of the 'text' string parameter.
                  Do Not use negative values here.

     height     - numeric, height of the item select box
                  code as zero to get default, the default height
                  is the text height of the font for the rastport.
                  Do Not use negative values here.

     flags      - numeric, flags for the item
                  Code as zero to get default, the default value
                  for menu-items and sub-items is
                  ITEMTEXT+HIGHCOMP+ITEMENABLED

                  If you specify a non-zero value here then you
                  must COMPLETELY specify the flag value.

                  eg. If you use COMMSEQ to specify that command
                  sequence character be displayed, then the flag
                  value is simply set to COMMSEQ.

                  the complete flag value should be specified as
                  ITEMTEXT+HIGHCOMP+ITEMENABLED+COMMSEQ
                  (or whatever you want)
```

```
        COM        - text string, command sequence character
                     the first character of the string is used

        fp         - numeric, front pen color for the item text

        bp         - numeric, back pen color for the item text

        dm         - numeric, draw mode to use for the item text

        itemfill   - pointer to an IntuiText, or Image structure
                     If this parameter is non-zero, then it will be
                     used, the IntuiText structure will NOT be built
                     from the text, fp, bp, and dm parameters.
                     If this parameter is -1 then APIG will leave the
                     menuitem's ItemFill parameter NULL.
                     Yes you can specify a pointer to an Image structure
                     here.

        selectfill - pointer to an Image structure
                     this Image will be displayed when the mouse pointer
                     is pointing to this sub-item. To use this the
                     HIGHIMAGE flag must be set.  If no image to display
                     then code this as a zero.  APIG uses whatever you
                     put here, do not use -1.

        Returns:   - pointer to sub-item

        Note that the menu is not attached to the window, the item
        is simply constructed.  You will need to call SetMenuStrip to
        actually attach the menu strip to the window.

        Also see menu hints.
```

1.57 MAKE

```
 <>  MAKESTRUCT(owner,type,size,mem_type)
    MAKEPOINTER is synonymous with MAKESTRUCT

    This function allocates an Intuition structure pointer.

    All pointers made by APIG have specific information about the
    pointer at negative offsets.  APIG uses this information to
    validate the pointer.  See the section below describing APIG
    pointers.
```

1.58 MAKETATTR

```
 <>  MAKETATTR(window,fontname,fontsize )

      Allocate and build an Text Font Attribute structure.
```

```
    The value returned by this function is used with OPENFONT() to load
    a text font from disk, and to specify fonts for intuitext.

    Note OPENFONT() returns a pointer to a TextFont structure, which
    is NOT used in MAKEITEXT().


    Inputs:

        window      - pointer to a window opened with OPENWINDOW().
                      This must be a pointer to a window.

        fontname    - string, name of the font to use

        fontsize    - numeric, size of the specified font to use


   Returns:         - pointer (rexx hex string) to the allocated TextAttr
                    - returns null ('0000 0000'x) if call fails
```

1.59 MENUNUMBER

```
 <>  MENUNUMBER(menustrip,menu,item,subitem)

    This function generates a 'MENUNUMBER'.

    Inputs:

        menustrip   - pointer to first menu in a menustrip

        menu        - pointer to a menu within the menustrip

        item        - pointer to a menu-item within the specified menu

        subitem     - pointer to a sub-item within the specified item
                      (code as zero if no sub-item)

    Returns:        - numeric, 'MENUNUMBER'.
```

1.60 MOUSEFREQUENCY

```
 <>  MOUSEFREQUENCY(window,N)

    This function allows you to specify how often you want to receive
    MOUSEMOVE messages from INTUITION.  The task managing the windows IDCMP
    builds an ARexx message for the MOUSEMOVE event it receives, and
    immediately replies to INTUITION, afterwhich it then sends the ARexx
    message packet to you.  The rate at which INTUITION sends MOUSEMOVEs
    might be faster than your ARexx macro can keep up with.  This function

    will cause the task to 'swallow' every 'Nth' MOUSEMOVE message.
```

```
Inputs:

    window    - pointer to window opened with OPENWINDOW().

    N         - numeric, frequency at which MOUSEMOVES should be
                sent to you

eg.  x = mousefrequency(window,50)
     will cause you to receive every 50th MOUSEMOVE message.

eg.  x = mousefrequency(window,1)
     will get you all MOUSEMOVE messages.

     specifing a value of zero will default to 1.

     When a window is initially opened its mousefrequency is 1.

Returns:      - returns N, the value you specified.

Note that you must specify MOUSEMOVES in the idcmp parameter in
order to get MOUSEMOVE event messages.
```

1.61 OPENWINDOW

```
 <>  OPENWINDOW( portname,left,top,wid,hgt,dpen,bpen,IDCMP,flags,
                          title,scr,console,bitmap,chkmark,gadlist


     Opens a window for use with the graphics/intuition library functions.

     Each call to this function creates a separate task to manage the IDCMP

     port for the window.

     Inputs:

         portname   - name of the public message port where IDCMP
                      messages should be sent. The public port
                      should be allocated prior to making the call
                      to 'OPENWINDOW'.

                      The parameter 'portname' is a string which is
                      the name of a public message port opened with
                      the ARexx OPENPORT function.

                      eg. portname = 'myport'
                          msgport  = openport(portname)

         left       - numeric, position of the left edge of the window

         top        - numeric, position of the top edge of the window

         wid        - numeric, width of the window

         hgt        - numeric, height of the window
```

```
          dpen         - numeric, detail pen to use

          bpen         - numeric, block pen to use

          IDCMP        - numeric, IDCMP flags - the set of messages that will
                         be reported.
                         The value you specify here determines what types of
                         messages you will receive from INTUITION.  Be careful
                         about the value you put here, since this value gets
                         passed to Intuition's OpenWindow() function.
                         eg. If you code a zero then Intuition doesnt even
                         bother opening up an IDCMP, and the window task will
                         never send you a message packet.
                         (youll then find yourself waiting forever for a
                          message to arrive at your Rexx message port)

          flags        - numeric, window flags

          title        - string, window title string

                         If the window is to appear in the workbench screen
                         code this as a zero

          console      - numeric, this parameter determines whether a
                         'console' will be attached to the window.
                         Specifying any non-zero value here will cause a
                         'console' to be attached.  You will then be able
                         to write to the window using WRITECONSOLE().

                         Note: Their is no function to read from the console
                               use the IDCMP RAWKEY or VANILLAKEYS flag.

          bitmap       - pointer to bitmap, if non-zero this implies that
                         the window is a superbitmap window.
                         'bitmap' should then point to a valid bitmap
                         obtained from MAKEBITMAP() function.  You must
                         also set the appropriate window/screen flag values.

          chkmark      - pointer to an image, if non-zero this should point
                         to an image obtained with the function LOADIMAGE().
                         This imagery will be used as the check mark for
                         menu items.

          gadlist      - pointer to a gadget, or list of gadgets that will be
                         placed in the window when the window opens. Obviously
                         these gadgets must be created before calling OPENWINDOW.
                         Code as zero if no gadget list.

     Only a few of the inputs are checked for validity, I can only assume
     you know what you want to do. So be careful when specifing the inputs.

     Returns:          - returns pointer to a window as a Rexx hex string.
                         returns null hex string (ie. '0000 0000'x) if open
                         fails.

      be found  the OPENWINDOW call will fail.
```

```
Intuition events are sent to the public port as RexxMsg packets.
Depending on the type of intuition event received, the argument
slots of the packet will contain specific information about the
event.

The intui-event information is returned in separate slots of the
ARexx message packet as follows:

     arg0 - CLASS
            numeric string, class of the intui-event

     arg1 - CODE
            numeric string, code of the intui-event

     arg2 - QUALIFIER
            numeric string, qualifier of the intui-event

     arg3 - MOUSEX
            numeric string, mouse position x coord.

     arg4 - MOUSEY
            numeric string, mouse position y coord.

     arg5 - SECONDS
            numeric string, event time stamp

     arg6 - MICROS
            numeric string, event time stamp

     arg7 - WINDOW
            a Rexx hex string, which is the address of the window
            in which the event occured.

     arg8 - IADDRESS
            a Rexx hex string, IAddress of the intui-event
            this typically will be the address of a gadget causing
            the event

     arg9 - GADGETID
            numeric string, if the event is a gadget related event
            this will be the GadgetID of the gadget

     The remaining slots are not used
GETPKT(), and GETARG() to examine the values of the intuition
message. After you are done using the message you should then
'reply' back to the window task using the ARexx function REPLY().

eg.  After allocating a port and opening a window;

    exitme = 0
    do forever  /* loop forever */

       x = waitpkt(portname)  /* wait for next message(s) to arrive */

       do forever    /* loop thru all received messages */
```

```
            msg = GETPKT(portname)
            if msg = '0000 0000'x then leave
            class      = GETARG(msg,0)
            code       = GETARG(msg,1)
            qualifier  = GETARG(msg,2)
            mousex     = GETARG(msg,3)
            mousey     = GETARG(msg,4)
            seconds    = GETARG(msg,5)
            micros     = GETARG(msg,6)
            window     = GETARG(msg,7)
            iaddress   = GETARG(msg,8)
            gadgetid   = GETARG(msg,9)  /* zero if not gadget related */

            if class = 512 then exitme = 1   /* class 512 = CLOSEWINDOW */

            x = REPLY(msg,0)    /* you MUST reply to all messages */
         end

         if exitme = 1 then leave

      end
```

the windows IDCMP port. This tasks creates several private msgports for communicating with the library functions, however one public message port is created for sending commands directly to the task. The name of this public message port will be the same as the task name (ie. 'apig.task.N'). Commands are sent to the public message port via the Rexx 'address' command.

The only commands recognized at the task public message port are

```
"DIE"  -  this command tells the task to kill itself and return all
          allocated resources it owns back to the system.
          Note the task knows nothing about independent structures
          that have been allocated and will NOT free them.
          It does know about all other memory that has been
          allocated to (owned by) its window and thus WILL free it.

          The primary use for this command will arise when the window
          is left hanging open due to termination of your ARexx macro
          because of an error.

          eg.  from CLI prompt you could enter
               (assuming you are running WSHELL)

          >    "address 'apig.task.1' DIE"

               This will (hopefully) tell the apig.task.1 to
               close the window and kill itself. If the window
               is opened on a custom screen, the screen will NOT
               be closed. This a change from version 1.1.

               If multiple windows are opened on a screen then
               each window must be told to die.

 Note:  If you know the address of the window you can also close it
        from the command line with:
```

```
> "say closewindow('wwww wwww'x)"
  where 'wwww wwww'x is the hex address of the window.
  if the window is on a screen, the screen will NOT be closed,
  even if the window is the last/only window in the screen.
  Memory (gadgets, intuitexts, menus, etc.) allocated to the
  window will be freed.

> "say closescreen('ssss ssss'x)"
   will close a screen from the command line

APIG will only close windows/screens which it has opened.
```

1.62 OPENSCREEN

```
 <>  OPENSCREEN( left,top,width,height,depth,dpen,bpen,vmodes,
                          type,title )

      Opens a custom screen for your use.
      Each call to this function will create a 'process', presently this
      process doesnt do much other than wait on signals from APIG.

      Inputs:

          left       - numeric, screens left edge

          top        - numeric, screens top edge

          width      - numeric, screens width

          height     - numeric, screens height

          depth      - numeric, screen depth, number of bit planes

          dpen       - numeric, detail pen to use

          bpen       - numeric, block pen to use

          vmodes     - numeric, view modes flag

          type       - numeric, type of screen
                       CUSTOMSCREEN, WBENCHSCREEN, SCREENBEHIND, etc.

          title      - string, screen title

     Returns:        - returns pointer to screen as a Rexx hex string
                       returns null hex string (ie. '0000 0000'x) if
                       open fails.

     The name of the process created will be 'apig.screen.N', where N
     is a sequential number to uniquely identify the screen process.
     A public port is available to send commands to the screen process.
     The name of this port is the same as the process name, ie.
     'apig.screen.N'.

     Now here is the dumb part, their are no commands for the screen process.
```

```
ANY message sent to the screen port, is an implied 'CLOSESCREEN'/kill
process.  (This may change)

         eg.   from CLI prompt you could enter
               (assuming you are running WSHELL)

           >    "address 'apig.screen.1' PITY THE FOOL"

           This will (hopefully) tell the apig.screen.1 to
           close the screen and kill itself.  Any memory
           allocated to ('owned' by) the screen will be freed.
           If any windows are open on the screen, the screen
           will NOT be closed.

The primary use for this command will arise when the screen
is left hanging open due to termination of your ARexx macro
because of an error.


Note:  If you know the address of the screen you can also close it
       from the command line with (assuming you have WShell):

       > "say closescreen('ssss ssss'x)"
         will close a screen from the command line

       APIG will only close windows/screens which it has opened.
```

1.63 PITEXT

```
<>  PITEXT(rp,left,top,text,fp,bp,dm,font)

     This function builds an IntuiText structure using the values
     you specify and then calls PrintIText to display the text in
     the RastPort.  The IntuiText structure created is immediately
     discarded.  The purpose is to allow you to render text in the
     rastport, without having to make SetAPen,SetBPen etc. calls
     between Text() calls.

     Inputs:

         rp         - pointer to rastport in which to display text

         left       - numeric, left edge offset relative to rastport

         top        - numeric, top edge offset relative to rastport

         text       - text string to be displayed

         fp         - numeric, front pen color

         bp         - numeric, back pen color

         dm         - numeric, draw mode (JAM1,JAM2,etc)

         font       - font to use, ( from MAKETATTR() )
```

```
Returns:       - always returns 1
```

1.64 SAVEIFF

```
<>  SAVEIFF(bitmap,filename,colortab,HAM,compress)

    This function uses Christian Weber's iff.library to save an IFF file.

    Inputs:

        bitmap     - pointer to bitmap containing IFF imagery.

        filename   - the name of the file to save IFF.

        colortab   - pointer to a color table, ie. array of 16bit values.

        HAM        - numeric, if non-zero specifies to save as HAM image.

        compress   - numeric, if non-zero specifies to save UN-COMPRESSED
                     default is to save COMPRESSED

    Returns:       - returns 1 if sucessful, otherwise zero.
```

1.65 SAVEIFFCLIP

```
<>  SAVEIFFCLIP(bitmap,filename,x,y,w,h,colortab,HAM,compress)

    This function uses Christian Weber's iff.library to save a
    rectangular region of a bitmap as an IFF file.

    Inputs:

        bitmap     - pointer to bitmap containing IFF imagery.

        filename   - the name of the file to save IFF.

        x          - numeric, horizontal offset into bitmap

        y          - numeric, vertical offset into bitmap

        w          - numeric, width of rectangle, expressed in bytes.
                     (ie. number of pixels divided by 8)

        h          - numeric, height of rectangle, expressed in lines.
                     (ie. number of raster lines)

        colortab   - pointer to a color table, ie. array of 16bit values.

        HAM        - numeric, only meaningful value here is a viewmode
                     of HAM (2048).
```

```
                       ie. a value of 2048  specifies to save as HAM image.
                       code as 2048 (0x0800) or zero

            compress   - numeric, if non-zero specifies to save UN-COMPRESSED
                       default is to save COMPRESSED

        Returns:       - returns 1 if sucessful, otherwise zero.
```

1.66 SCRPROCNAME

```
 <>  SCRPROCNAME(scrptr)

    This function returns the name of the process for the screen.
    This is also the name of the public message port for the screen.

      Inputs:

          scrptr  - pointer to a screen opened with OPENSCREEN().

      Returns:    - screen process name string, ie. 'apig.screen.N'
```

1.67 SET_APIG_GLOBALS

```
 <>  SET_APIG_GLOBALS()

    This function initializes many of the global constants used by/in the
    various Intuition/Graphic data structures and makes them available to
    your ARexx macro.  Calling this function will modify your ARexx macro
    environment so that the variables listed below are defined and valued.
    Of course you could just as well make assignments in your ARexx macro.
    (any value not listed below you will have to assign in your ARexx pgm)

    Inputs:   - none

    Also be aware that the use of the 'procedure' keyword in an internal
    function creates a new symbol table, these variables will not be
    defined within the procedure unless you also call SET_APIG_GLOBALS
    again from within the procedure.

    SET_APIG_GLOBALS creates and sets the following variables:

    *** See file set_apig_globals.txt ***

    eg. after calling SET_APIG_GLOBALS

    say "Lace = " lace

    should display

    Lace = 4
```

1.68 SETARRAY

```
 <>  SETARRAY( arrayptr, arrayindx, value )

      This function allows you to store 16-bit values into the array
      pointed to by 'arrayptr'.  Primarily intended for initializing
      arrays of 16bit integers for functions like LOADRGB(), POLYDRAW(),
      and MAKEBORDER().

      Inputs:

          arrayptr   - pointer to array (block of memory) as returned
                       by ARexx ALLOCMEM().  The array should be allocated
                       in multiples of 2bytes.

          arrayindx  - numeric, index position you wish to set
                       (index starts from zero)

          value      - numeric, value to set position to

     Returns:        - returns the value the array position was set to.

     Also See: SETX(), SETY(), GETX(), GETY(), GETARRAY()
```

1.69 SETNEWGADGET

```
 <> SETNEWGADGET(ngad,vinfo,font,left,top,width,height,text,flags,id,usrdata)

      Modifies the values in a previously allocated NewGadget structure.

      Inputs:

      ngad      - pointer to NewGadget structure created with
                  MAKENEWGADGET().

      vinfo     - pointer to visual info, obtained from prior call
                  to GETVISUALINFO()

      font      - pointer to font to be used for gadget.

      left      - numeric, left edge placement of gadget

      top       - numeric, top edge placement of gadget

      width     - numeric, width of gadget

      height    - numeric, height of gadget

      text      - string, text to be used for gadget label

      flags     - numeric specifies placement of gadget label text
                  (PLACETEXT_LEFT/PLACETEXT_IN/PLACETEXT_ABOVE/etc.)

      id        - numeric, value to be used for gadget ID
```

```
usrdata   - pointer, to whatever you want to point to.

Returns   - 1 if successful, otherwise 0.

NOTE: A buffer (ng_GadgetText) is allocated to hold the 'text' string
      supplied, this buffer is 'owned' by the NewGadget structure, and
      will be freed when the NewGadget structure is freed.
```

1.70 SETSELECT

```
 <>  SETSELECT(gadgetptr,state)

      This function sets the gadet select state.

      Inputs:

      gadgetptr - pointer to a boolean gadget

      state     - numeric, a non-zero value sets the gadet select flag,
                  zero clears the gadget select flag.

      Returns:  - always returns 1
```

1.71 SETGADTYPE

```
 <>  SETGADTYPE( gadgetptr,gadtype )

      This function modifies the value of the GadgetType field for the
      specified gadget.  Note the gadtype value is 'ORed' with the current
      value of GadgetType for the gadget.  Specifically meant for creating
      GZZGADGETs, but any valid gadgetype(s) can be used.


      Inputs:

          gadgetptr  - pointer to a gadget

          gadtype    - numeric, value


     Returns:        - returns 1 if type was modified, else returns 0.
```

1.72 SETIMAGE

```
 <>  SETIMAGE(image,left,top,ppick,ponoff)

      This function modifies the variables of an Image structure.
      This allows you to link Images together, so that when they are
      drawn they are offset by dx, dy. Otherwise the images would be
      displayed one on top of each other.

      Inputs:

          image      - pointer to an image structure

          left       - numeric, value to set leftedge
                       use -1 if value is not to be modified

          top        - numeric, value to set topedge
                       use -1 if value is not to be modified

          ppick      - numeric, value to set PlanePick
                       use -1 if value is not to be modified

          ponoff     - numeric, value to set PlaneOnOff
                       use -1 if value is not to be modified


      Returns:       - always returns 1
```

1.73 SETSTRGAD

```
 <>  SETSTRGAD( gadgetptr,text )

      This function sets the value of the string gadget to 'text'
      You must refresh the gadget to see the new string.

      Inputs:

          gadgetptr  - pointer to a string gadget

          text       - text string to be stored in the string gadget
                       Note to clear a string gadget use a empty string
                       using double quotes.

                       eg.  x = SETSTRGAD(mygad,"")  THIS

                            x = SETSTRGAD(mygad,0)   NOT this
                                this puts the character '0' in the gadget

     Returns:        - returns the length of the string installed in the
                       string gadget buffer.  This should be the length
                       of the 'text' you specified.  If the length of
                       the 'text' is greater than the max chars the string
                       gadget buffer can hold, then (maxchars - 1) are
                       placed in the buffer.
```

1.74 SETSTRGADID

```
 <>  SETSTRGADID( window,gadid,text,requester )
```

```
     This function is similar to SETSTRGAD, only it uses a GadgetID
     instead of a gadget pointer.

     Inputs:

         window     - pointer, gadgets window

         gadid      - numeric, gadget's ID

         text       - text string to be stored in the string gadget

         requester  - pointer, pointer to requester if in a requester
                      else make this null.

    Returns:        - returns the length of the string installed in the
                      string gadget buffer.  This should be the length
                      of the 'text' you specified.  If the length of
                      the 'text' is greater than the max chars the string
                      gadget buffer can hold, then (maxchars - 1) are
                      placed in the buffer.
```

1.75 SETTAGSLOT

```
 <>  SETTAGSLOT(tagarray,slot,tag,'p'/'n',value)

     This function is similar to SETVALUE, but is SPECIFIALLY intended
     for setting the values in a tagarray/taglist.  Both the 'tag' and
     the 'tagdata' values are set.

     tagarray - pointer, to taglist (block of mem)

     slot     - numeric, specifies the relative position, from the beginning
                of the tag array

     tag      - pointer, tagvalue, APIG expects ALL TAGS to be expressed
                in hex-string form. (ie. '80080035'x)

     type     - string, either a 'N' or 'P' this specifies the form
                of the data 'value' you are passing.

                'N' specifies that the value is a numeric.
                'P' specifies that the value you are passing is a pointer.
                (ie. ARexx hex string)


     value    - numeric/pointer, the data value

     Note that no size is specified the tag value and tagdata value are
     always considered as 4bytes each.  Therefore the allocation of your
     tagarray should ALWAYS be in multiples of 8bytes.

     returns -  always returns 1

     eg.
          mytagarray = ALLOCMEM(24,MEMF_CLEAR)  /* 24 bytes */
```

```
        mypointer  = ALLOCMEM(120,MEMF_CLEAR)
        call SETTAGSLOT(mytagarray,0,'80000092'x,'n',35)
        call SETTAGSLOT(mytagarray,1,'80000077'x,'p',mypointer)
        call SETTAGSLOT(mytagarray,2,TAG_DONE,'n',0)

  byteoffset 0  = mytagarray[0] = '80000092'x  <---+ (slot 0)
             4  = mytagarray[1] = 35

             8  = mytagarray[2] = '80000077'x  <---+ (slot 1)
             12 = mytagarray[3] = mypointer

             16 = mytagarray[4] = TAG_DONE     <---+ (slot 2)
             20 = mytagarray[5] = 0
```

1.76 SETVALUE

```
 <>  SETVALUE(ptr,offset,size,type,value,len)

      This function allows you to set the value of any parameter in
      any data structure.  Please be very certain about the parameters you
      use in this function. (If you're gonna shoot yourself in the foot
      this is the gun to do it with.)

      ptr    - pointer, (ARexx hex string) to any data structure, ie. window
               screen, bitmap, etc.

      offset - numeric, specifies the relative position, from the beginning
               of the data structure, of the data value you want to set.
               (see RKM or include '.i' files for offsets)

      size   - numeric, specifies the size of the data value you want to
               set.  This value must be either 1, 2, or 4.  Any other
               value will cause the function to return a NULL.
               (ie. '0000 0000'x).

      type   - string, either a 'N', 'P' or 'S', this specifies the form
               of the data 'value' you are passing.

               'N' specifies that the value is a numeric.
                   for sizes of 1 and 2 the value is always taken (assumed)
                   to be a numeric.

               If the size is 4 then you can also use (in addition to 'N')
               the following:

               'P' specifies that the value you are passing is a pointer.
                    (ie. ARexx hex string)

               'S' specifies that the value you are passing is a string.
                   VERY IMPORTANT NOTE !!!
                   When using 'P' or 'S',  'ptr' + 'offset' must result
                   in an address which contains a pointer to something.

      value  - numeric/string/pointer, the value to be used
      len    - numeric, used in conjuction with type 'S', this specifies
               the maximum number of characters to be moved into the
               area pointed to by the pointer at 'ptr' + 'offset'.
               If you specify -1 the entire string will be copied into the
               area pointed to by the pointer.
               eg.  You can set the window title string with:

                    x = setvalue(windowpointer,32,4,'S',"MY NEW TITLE",-1)

                    This statement DIRECTLY modifies the contents of the
                    buffer pointed to by the window title pointer.

              The proper thing to do would be to modify where the window
              title pointer points.
              eg.
                  mynewtitle = ALLOCMEM(length("MY NEW TITLE")+1,'0001 0000'x)
                  call EXPORT(mynewtitle,"MY NEW TITLE")
                  x = setvalue(windowpointer,32,4,'P',mynewtitle,0)

      Returns -  1 if succesful, otherwise 0
```

1.77 SETX

```
 <>   SETX( arrayptr,xindex,value )

      This function does the same as SETARRAY(), the difference is that
      it computes the index offset value for X-pair for you.

      Inputs:

          arrayptr   - pointer to array as returned by ARexx ALLOCMEM().

          xindex     - numeric, index position you wish to set

          value      - numeric, value to set position to

     Returns:        - returns the value the array position was set to.
```

1.78 SETY

```
 <>   SETY( arrayptr,yindex,value )

      This function does the same as SETARRAY(), the difference is that
      it computes the index offset value for Y-pair for you.

      Inputs:

          arrayptr   - pointer to array as returned by ARexx ALLOCMEM().

          yindex     - numeric, index position you wish to set

          value      - numeric, value to set position to
```

```
    Returns:        - returns the value the array position was set to.
```

1.79 VERT

```
 <>  VERTPOT(propgadgetptr) <>  VERTBODY(propgadgetptr)

      These functions return the value of the vertical components of the
      pot gadget.

      Inputs:

      propgadgetptr  - pointer to a proportional gadget, return from
                       MAKEPROPGADGET().

     Returns:        - returns the numeric value of VertPot
```

1.80 TICKFREQUENCY

```
 <>  TICKFREQUENCY(window,N)

      This function allows you to specify how often you want to receive
      INTUITICK messages from INTUITION.  The task managing the windows
      IDCMP builds an ARexx message for the INTUITICK event it receives,
      and immediately replies to INTUITION, afterwhich it then sends the
      ARexx message packet to you.  The rate at which INTUITION sends
      INTUITICKS is about 10 times per sec, which might be faster than
      your ARexx macro can keep up with.  This function will cause the
      task to 'swallow' every 'Nth' INTUITICK message.

      Inputs:

          window    - pointer to window opened with OPENWINDOW().

          N         - frequency at which INTUITICKS should be sent to you

      eg.  x = tickfrequency(window,50)
           will cause you to receive every 50th INTUITICK message.
           (approx. one every 5 seconds)

      eg.  x = tickfrequency(window,1)
           will get you all INTUITICK messages.

           specifing a value of zero will default to 1.

           When a window is initially opened its tickfrequency is 20.

      Returns:      - returns N, the value you specified.
```

1.81 USEIFFCOLOR

```
<>  USEIFFCOLOR(pointer,scr)

      This functions sets the screens color registers to the colors
      contained in the 'CMAP' chunk of the IFF.  If the IFF has no
      CMAP then no change is made to the screen colors.

      Inputs:

          pointer    - pointer to a IFF bitmap (returned by LOADIFF()).
                       This must be the original bitmap pointer
                       (or copy of it) returned by the LOADIFF() function.
                       You cannot use a pointer returned by MAKEBITMAP()
                       into which you have blitted the IFF.

          scr        - pointer to a screen opened with OPENSCREEN()

      Returns:       - does a LOADRGB() of the IFF colors into the screen
                       and returns 1 if successful.
                       returns 0 if not a pointer to IFF or if the IFF
                       does not contain a CMAP chunk.
```

1.82 WINDOWINFO

```
<>  WINDOWINFO(window,code)

      This function returns various values from the window structure.

      Inputs:

          window   - pointer to window opened with OPENWINDOW().

          code     - numeric, code which specifies which value from the
                     window structure you would like.

                     The code values are:

                       CODE       Returns
                     ---------  ---------------------------
                        1         Windows Left Edge value
                        2         Windows Top Edge value
                        3         Windows Width  value
                        4         Windows Height value
                        5         Windows MouseY value
                        6         Windows MouseX value
                        7         Windows MinWidth value
                        8         Windows MinHeight value
                        9         Windows MaxWidth value
                        10        Windows MaxHeight value
                        11        Windows Flags value
                        12        Windows Requester count value
                        13        Windows Border Left value
                        14        Windows Border Top value
                        15        Windows Border Right value
                        16        Windows Border Bottom value
```

17	Windows IDCMP Flags value
18	Windows Detail Pen value
19	Windows Block Pen value
20	Windows GZZMouseX value
21	Windows GZZMouseY value
22	Windows GZZWidth value
23	Windows GZZHeight value

1.83 WINTASKNAME

```
<>  WINTASKNAME(window)

   This function returns the name of the task managing the windows
   IDCMP port.  This is also the name of the public message port
   for the window.

     Inputs:

         window  - pointer to a window opened with OPENWINDOW().

     Returns:    - window task name string, ie. 'apig.task.N'
```

1.84 WRITECONSOLE

```
<>  WRITECONSOLE(window,text)

     This function writes the text to console of the window.
     The window must have been opened with a console attached.

     Inputs:

         window   - pointer to window opened with OPENWINDOW().

         text     - text string to be written to console.

     Returns:     - the number of characters written to the console.
```

1.85 Graphics Library functions

```
                         Graphics Library Functions


 After parameter conversion the following functions result in a
 direct call to the Amiga Graphic Library function of the same
 name.  Please refer to your favorite Amiga reference book(s) for
 a detailed description.
```

```
<>  AREACIRCLE(rp,cx,cy,radius)

<>  AREADRAW(rp,x,y)

<>  AREAELLIPSE(rp,cx,cy,a,b)

<>  AREAEND(rp)

<>  AREAMOVE(rp,x,y)

<>  BITMAPSCALE(bitscaleargs)

<>  BLTBITMAP(srcbm,srcx,srcy,dstbm,dstx,dsty,sizex,sizey,minterm,mask,tempa)

<>  BLTBITMAPRASTPORT(srcbm,srcx,srcy,rp,destx,desty,sizex,sizey,minterm)

<>  BLTCLEAR(memblock,bytecount,flags)

<>  BLTMASKBITMAPRASTPORT(scrbm,srcx,srcy,rp,destx,desty,sizex,sizey,minterm,bltm
    ask)

<>  BLTPATTERN(rp,mask,x1,y1,x2,y2,bytecnt)

<>  BLTTEMPLATE(srctemplate,srcx,srcmod,rp,dstx,dsty,sizex,sizey)


<>  CLEAREOL(rp,x,y)

        Note:
        The Amiga Graphics library function has one argument.
        X,Y (starting pixel position) are included in this
        call to mark where the clearing should start from.

<>  CLEARSCREEN(rp,x,y)

        Note:
        The Amiga Graphics library function has one argument.
        X,Y are included in this call to mark where the
        clearing should start from.

<>  CLIPBLIT(srcrp,srcx,srcy,destrp,destx,desty,xsize,ysize,minterm)

<>  CLOSEFONT(font)

<>  CLOSEMONITOR(monitor_spec)

<>  DRAW(rp,x,y)

<>  DRAWCIRCLE(rp,cx,cy,r)

<>  DRAWELLIPSE(rp,cx,cy,a,b)

<>  FINDDISPLAYINFO(id)

<>  FLOOD(rp,mode,x,y)

<>  FONTEXTENT(font,fontextent)
<>  GETDISPLAYINOFDATA(handle,buf,size,tag,id)

<>  GETVPMODEID(viewport)

<>  INITAREA - see MAKEAREA function

<>  INITBITMAP - see MAKEBITMAP function

<>  INITRASTPORT - see MAKERASTPORT function

<>  LOADRGB4(screen,arrayptr,count)

<>  MODENOTAVAILABLE(id)

<>  MOVE(rp,x,y)

<>  NEXTDISPLAYINFO(id)

<>  OPENFONT(textAttr)
     Note: textAttr is obtained from MAKETATTR()

<>  OPENMONITOR(monitor_name,display_id)
     Note: monitor_name = stringvar/literal

<>  POLYDRAW(rp,count,array)

<>  READPIXEL(rp,x,y)

<>  RECTFILL(rp,xmin,ymin,xmax,ymax)

<>  SCALERDIV(factor,numerator,denominator)

<>  SCROLLRASTER(rp,dx,dy,xmin,ymin,xmax,ymax)

<>  SETAFPT(rp,pattern,patternsize)

<>  SETAPEN(rp,pen)

<>  SETBPEN(rp,pen)

<>  SETDRMD(rp,mode)

<>  SETDRPT(rp,linepattern)

<>  SETFONT(rp,font)

<>  SETOPEN(rp,pen)

<>  SETRAST(rp,pen)

<>  SETRGB4(screen/window,pen,r,g,b)
      Note:
      This function does the same as SETRGB4() but the parameters are
      different. The first parameter is a pointer to a screen or window.
```

```
<>  SETSOFTSTYLE(rp,style,enable)

<>  SETWRMSK(rp,wrtmask)

<>  TEXT(rp,string,count)
      Note:
      If the count parameter is less than zero, then the
      length of the string will be computed for you.
      eg.  call TEXT(rp,string,-1), the length of string is computed for you
           (string must be null terminated)

<>  TEXTEXTENT(rp,string,count,textextent)

<>  TEXTFIT(rp,string,len,textextent,consextent,strdir,consbitwid,consbithgt)

<>  TEXTLENGTH(rp,string,count)
      Note:
      If the count parameter is less than zero, then the
      length of the string will be computed for you.
      (string must be null terminated)


<>  WRITEPIXEL(rp,x,y)
```

1.86 Intuition library functions

```
                         Intuition Library Functions


 After parameter conversion the following functions result in a
 direct call to the Amiga Intuition Library function of the same
 name.  Please refer to your favorite Amiga reference book(s) for
 a detailed description.


<>  ACTIVATEGADGET(gadget,window,requester)

<>  ACTIVATEWINDOW(window)

<>  ADDGADGET(window,gadget,position)

<>  ADDGLIST(window,gadget,position,numgad,requester)

<>  AUTOREQUEST(window,itext,itext,itext,posflags,newflags,width,hgt)

      Note: This function does not return until the requester is
            satisfied.  Only one requester can be put up at a time.


<>  BEGINREFRESH(window)

<>  BUILDEASYREQUESTARGS(window,easystruct,idcmp,args)

<>  CHANGEWINDOWBOX(window,left,top,width,height)
<>  CLEARDMREQUEST(window)

<>  CLEARMENUSTRIP(window)

      Note: APIG library will call CLEARMENUSTRIP() before
            closing a window.  The library will also call
            CLEARMENUSTRIP() before setting a new menu strip.
            You will need this function however if you set a
            menustrip, but then later want to remove the menu
            without setting a new one. (ie. no menus displayed)


<>  CLEARPOINTER(window)

<>  CLOSESCREEN(screen)
     APIG will refuse to close a screen if a window is still open.
     APIG will refuse to close a screen that it did not open.

<>  CLOSEWINDOW(window)
     APIG will refuse to close a window that it did not open.

<>  DISPLAYBEEP(screen)

<>  DISPOSEOBJECT(object)
       object = ptr returned from NEWOBJECTA()

<>  DOUBLECLICK(startsecs,startmicros,currentsecs,currentmicros)

<>  DRAWBORDER(rp,border,leftoffset,topoffset)

<>  DRAWIMAGE(rp,image,leftoffset,topoffset)

<>  DRAWIMAGESTATE(rp,image,leftoffset,topoffset,state,drinfo)

<>  EASYREQUEST(window,title,bodytext,gadtext,arglist,IDCMP,flags)

    Builds an EasyStruct structure and passes it to EasyRequestArgs().

    Inputs:

      window   - pointer, to window

      title    - string, placed in requester title bar

      bodytext - string, placed in body of requester.  May contain 'C' style
                 formatting strings.

      gadtext  - string, specifies gadgets in requester, separated with
                 a vertical bar ( '|' ). May contain 'C' style formatting
                 strings.

      arglist  - pointer, to list of data values for bodytext and gadtext
                 arguments.

      IDCMP    - numeric, set of IDCMP flags which will satisfy requester
                 (not a pointer)
      flags    - numeric, for whatever future use C= has for EasyRequest flags
                 this argument is faithfully moved into EasyStruct.

      Returns: numeric, after requester is satisfied the result will be
               a positive or negative number. If the requester was satisfied
               by clicking on one of the gadgets then the returned value is
               the number of gadget clicked. Numbered from left to right
               per RKM as: 1, 2, ..., N, 0.  Where 0 is for the rightmost
               gadget in the requester.
               If the requester was satisfied by an IDCMP event then the
               returned value will be the negative value of the IDCMP
               class.  ie. returns -DISKINSERTED for DISKINSERTED.

      NOTE: Only allows one EasyRequest to be put up at a time.


<>  EASYREQUESTARGS(window,es,IDCMP_ptr,ArgList)

    All arguments are as per RKM.

    window    - pointer (hex-string)
    es        - pointer
    IDCMP_ptr - pointer (not a numeric)
    ArgList   - pointer

    You will have to build the EasyStruct ('es') the hard way with
    SETVALUE().

    Return values are as per RKM, (1,2,...N,0) for gadgets, -1 for IDCMP.
    If -1 is returned the IDCMP value is stored at IDCMP_ptr, use GETVALUE
    to retrieve it.

<>  ENDREFRESH(window,complete)

<>  ENDREQUEST(requester,window)

<>  FREESCREENDRAWINFO(drinfo)

<>  FREESYSREQUEST(window)

<>  GETATTR(attrid,object,storageptr)

<>  GETDEFPREFS(prefbuffer,size)

<>  GETDEFAULTPUBSCREEN(namebuffer)

<>  GETPREFS(prefbuffer,size)

<>  GETSCREENDATA(buffer,size,type,screen)

<>  GETSCREENDRAWINFO(screen)

<>  ITEMADDRESS(menustrip,menunumber)

<>  ITEMNUM(menunumber)

<>  INITREQUESTER - see MAKEREQUESTER
<>  INTUITEXTLENGTH(itext)

<>  LOCKPUBSCREEN(screename/null)

<>  LOCKPUBSCREENLIST()

<>  MENUNUM(menunumber)

<>  MODIFYIDCMP(window,idcmpflags)

<>  MODIFYPROP(gadget,window,requester,flags,hpot,vpot,hbody,vbody)

<>  NEWMODIFYPROP(gadget,window,requester,flags,hpot,vpot,hbody,vbody,
                                                             numgad)

<>  NEWOBJECTA(class,classid,taglist)
      NOTE:  best if class = null(), APIG will accept whatever you put here
             classid = stringvar/literal, NOT a hex-string pointer
             taglist = pointer to taglist (hex-string)

<>  NEXTPUBSCREEN(screen,namebuffer)

<>  MOVESCREEN(screen,deltax,deltay)

<>  MOVEWINDOW(window,deltax,deltay)

<>  MOVEWINDOWINFRONTOF(window,behindwindow)

<>  OFFGADGET(gadget,window,requester)

<>  OFFMENU(window,menunumber)

<>  ONGADGET(gadget,window,requester)

<>  ONMENU(window,menunumber)

<>  OPENSCREENTAGLIST(newscreen,taglist)
    Note: Will create a new ’process’ like OpenScreen().

<>  OPENWINDOWTAGLIST(portname,newwindow,taglist,console)

    Like OPENWINDOW(), OPENWINDOWTAGLIST() needs a portname.
    The parm ’console’ specifies if a console is to be attached.

    Note: If you are going to use a NewWindow or NewScreen structure,
          then you will have to "hardcode" them using SETVALUE().
          Probably better to leave it as null(), and pass specs through
          the taglist. Also, OPENSCREENTAGS() and OPENWINDOWTAGS() are not
          implemented in APIG.


<>  PRINTITEXT(rp,itext,leftoffset,topoffset)

<>  PUBSCREENSTATUS(screen,statusflags)

<>  REFRESHGADGETS(gadgets,window,requester)
<>   REFRESHGLIST(gadgets,window,requester,numgad)

<>   REFRESHWINDOWFRAME(window)

<>   REMOVEGADGET(window,gadget)

<>   REMOVEGLIST(window,gadget,numgad)

<>   REPORTMOUSE(boolean,window)

<>   RESETMENUSTRIP(window,menu)

<>   REQUEST(requester,window)

       This function unlike the AUTOREQUEST() function, returns immediately.
       Multiple requesters can be put up.

       This function will return 1 if the requester was successfully put
       up in the window, else it returns 0.

       If successful you should then enter a loop to receive messages from
       the window task.  The IDCMP of the window will have been modified,
       you will get your normal IDCMP messages in addition to GADGETDOWN,
       GADGETUP, REQCLEAR and REQSET messages.
       When the requester is satisfied, the IDCMP will be restored.

       eg.  after building a requester

            reqputup = request(myrequester,mywindow)
            if reqputup = 1 then   /* if result is 1 request successful */
               do                  /* else requester was not put up     */
                   exitme = 0
                   do forever

                      if exitme = 1 then leave

                      x = waitpkt(portname)
                      do forever
                         msg = getpkt(portname)
                         if msg = '0000 0000'x then leave
                         class  = getarg(msg,0)

                         if class = REQCLEAR then exitme = 1

                         if class = GADGETDOWN then
                            do
                               gadid = getarg(msg,9)
                               if gadid = mygadgetid1 then
                               do
                                  x = dosomething1()
                                  exitme = 1
                               end
                            end
                         if class = GADGETUP then
                            do
                               gadid = getarg(msg,9)
```

```
                              if gadid = mygadgetid2 then
                              do
                                 x = dosomething2()
                                 exitme = 1
                              end
                           end

                        x = replymsg(msg,0)
                     end
                  end
              end
```

```
<>   SCREENTOBACK(screen)

<>   SCREENTOFRONT(screen)

<>   SETDEFAULTPUBSCREEN()

<>   SETDMREQUEST(window,dmrequester)

<>   SETMENUSTRIP(window,menu)

      Note: APIG library keeps track of whether a menu
            strip has been attached to a window.  Therefore
            you can call SETMENUSTRIP() multiple times without
            having to precede it with a CLEARMENUSTRIP() call.
            The library will call CLEARMENUSTRIP() for you if
            it sees that a menu is still attached to the window.
            It will also call CLEARMENUSTRIP() before closing
            a window as well.


<>   SETMOUSEQUEUE(window,newlength)

<>   SETPOINTER(window,pointer,height,width,xoffset,yoffset)

<>   SETPREFS(prefbuffer,size,inform)

<>   SETPUBSCREENMODES(modes)

<>   SETWINDOWTITLE(window,widnowtitle,screentitle)

<>   SHOWTITLE(screen,showit)

<>   SIZEWINDOW(window,deltax,deltay)

<>   SUBNUM(menunumber)

<>   SYSREQHANDLER(window,idcmpflagsptr,waitinput)

<>   UNLOCKPUBSCREEN(screenname/null,screenptr)

<>   VIEWADDRESS()

<>   VIEWPORTADDRESS(window)
<>   WBENCHTOBACK()

<>   WBENCHTOFRONT()

<>   WINDOWLIMITS(window,minwidth,minheight,maxwidth,maxheight)

<>   WINDOWTOBACK(window)

<>   WINDOWTOFRONT(window)

<>   ZIPWINDOW(window)
```

1.87 layers library functions

```
                         Layers Library Functions

 After parameter conversion the following functions result in a
 direct call to the Amiga Layers Library function of the same name.
 Please refer to your favorite Amiga reference book(s) for a detailed
 description.



<>   BEHINDLAYER(layer)
       Note that only one parameter is used.
       layer - is a pointer to a layer structure


<>   CREATEBEHINDLAYER(windowpointer,x0,y0,x1,y1,flags,bm2)
       Note that a window pointer (obtained from OPENWINDOW()) is used
       instead of the parameters layerinfo and bitmap as described in
       the RKM manual.
       The layerinfo and bitmap pointers are obtained from the window
       parameter, all other parameters are as described in RKM.


<>   CREATEUPFRONTLAYER(windowpointer,x0,y0,x1,y1,flags,bm2)
       Note that a window pointer (obtained from OPENWINDOW()) is used
       instead of the parameters layerinfo and bitmap as described in
       the RKM manual.
       The layerinfo and bitmap pointers are obtained from the window
       parameter, all other paramenters are as described in RKM.


<>   DELETELAYER(layer)
       Note that only one parameter is used.
       layer - is a pointer to a layer structure


<>   MOVELAYERINFRONTOF(layertomove,targetlayer)


<>   MOVELAYER(layer,dx,dy)
       Note that only three parameters are used.
```

```
      layer - is a pointer to a layer structure


<>   SCROLLLAYER(layer,dx,dy)
      Note that only three parameters are used.
      layer - is a pointer to a layer structure


<>   SIZELAYER(layer,dx,dy)
      Note that only three parameters are used.
      layer - is a pointer to a layer structure


<>   UPFRONTLAYER(layer)
      Note that only one parameter is used.
      layer - is a pointer to a layer structure
```

1.88 ASL library functions

```
                         ASL Library Functions


 After parameter conversion the following functions result in a
 direct call to the Amiga ASL Library function of the same name.
 Please refer to your favorite Amiga reference book(s) for a
 detailed description.



<>   ALLOCASLREQUEST(type,tags)

       Note you must use FREEASLREQUEST/FREEFILEREQ and not one of APIG's
       FREE...() functions. However the returned requester can be an
       owner.

<>   ALLOCFILEREQUEST()

       Note you must use FREEASLREQUEST/FREEFILEREQ and not one of APIG's
       FREE...() functions. However the returned requester can be an
       owner.


<>   ASLREQUEST(req,tags,owner)

    This function differs slightly from the Amiga ASL library function, in
    that it has a third optional parameter, the return value is different
    as well.

    Inputs:
      req    - per RKM
      tags   - per RKM

      owner  - pointer, applies only to FONT requesters.
               A pointer to ANY owner object that will 'own'
               the returned value.
```

```
returns - if the requester, 'req', is a ASL_FILE requester then the
          returned value is string which is the name of the file
          selected, or a null hex-string if no file selected.

          if the requester, 'req', is a ASL_FONT requester then the
          returned value is a hex-string pointer, which points to
          a TextAttr structure of the selected font. It is a copy
          of the TextAttr struct in the FontRequester not a pointer
          to it.  If no font is selected then a null hex-string is
          returned.

          Since the returned TextAttr is allocated memory it must
          be freed at some point. If 'owner' is specified then it
          will be freed when its owner is freed. If 'owner' is not
          specified, then the returned TextAttr will be an independent
          structure and must be explictly freed with FREETHIS().

          Be aware that if you do not specify an owner, then each
          fontrequest call to ASLREQUEST() will allocate memory for
          the TextAttr.

          Thus,
                do 1 to 5
                   x = ASLREQUEST(req,tags)
                end

          Will allocate five TextAttr's, each overlaying the previous
          'x' value and therefore you will only free the last one
          allocated.

          You could do this:

                do 1 to 5
                   x = ASLREQUEST(req,tags)
                        .
                        .
                        .
                   call FREETHIS(x)
                end


          Or if you are gonna keep the requester around, then
             do 1 to 5
                x = ASLREQUEST(req,tags,req)
             end
             call FREEASLREQUEST(req)

          Will allocate the TextAttr to the requester 'req'
          when you free the requester all of the TextAttr's
          will be freed also. (Any owner will do)

          The returned TextAttr can be used where a TextAttr is
          needed, IntuiText or the PITEXT() function for example.
```

```
<>  FREEASLREQUEST(filerequest)

<>  FREEFILEREQ(filerequest)

<>  REQUESTFILE(req,multi,save,hail,dir,file,pat,nofile,win,left,top
            ,width,hgt,sep)

       Note the Amiga ASL library function has one argument, 'req', you
       can specify optional arguments in APIG's REQUESTFILE() call in order
       to modify the ASL defaults. The returned value is also different
       from the Amiga ASL function.

       This function returns after the user has selected either the "OK"
       or "CANCEL" buttons.

       If "CANCEL" is selected the returned value is a null hex-string.

       If "OK" is selected the returned value is the directory/filename
       string of the selected file(s).


       Optional arguments are:

       multi  - numeric, any non-zero value specifies multiple selection

       save   - numeric, any non-zero value specifies request is a save
                operation.

       hail   - string, prompt string placed in requester title bar.

       dir    - string, initial directory path

       file   - string, initial filename

       pat    - any non-zero value will cause a pattern gadget to be
                included in the requester.

       nofile - any non-zero value will cause only directories to be
                displayed in the requester.

       win    - pointer, to window in which the requester will be displayed.

       left   - numeric, left edge placement

       top    - numeric, top edge placement

       width  - numeric, width of requester

       hgt    - numeric, height of requester

       sep    - string, used to separate files when multiple selections
                are made. The default separator character is a '|'.
                The entire string is used, not just the first character.

       NOTE: Optional arguments are positional use zero or null() where
             appropriate to maintain positionality. If any optional
             arguments are specified, then APIG builds a taglist and
             calls LVOAslRequest. If only 'req' is specified then
             LVORequestFile is called.
```

1.89 Exec library functions

```
                        Exec Library Functions


 After parameter conversion the following functions result in a
 direct call to the Amiga Exec Library function of the same name.
 Please refer to your favorite Amiga reference book(s) for a
 detailed description.


     When using the list functions, parameters are allocated from
     system memory using the ARexx function ALLOCMEM().  Be sure
     that you allocate at least as much memory required for the list
     or node data structure.

<>   ADDHEAD(list,node)

<>   ADDTAIL(list,node)

<>   ALLOCVEC(size,type)

<>   ENQUEUE(list,node)

<>   FREEVEC(vecptr)

<>   INSERTNODE(list,node,listnode)
        Note name differs, since ARexx has function with the name INSERT().

<>   NEWLIST(list)

<>   REMHEAD(list)

<>   REMOVE(node)

<>   REMTAIL(list)
```

1.90 APIG list related functions

```
                  APIG Library List related functions.


    You may find these functions useful when working with Exec list
    structures.


<>   EMPTYLIST(list) - returns 1, if the list is empty, else returns 0

<>   LISTEMPTY(list) - same as EMPTYLIST()
<>  FIRSTNODE(list,node) - returns 1 if the node is first node in the list
                           else returns 0.

<>  LASTNODE(list,node)  - returns 1 if the node is the last node in the list
                           else returns 0.
```

1.91 Gadtools library functions

```
                        GadTools Library Functions


 After parameter conversion the following functions result in a
 direct call to the Amiga GadTools Library function of the same name.
 Please refer to your favorite Amiga reference book(s) for a
 detailed description.


<>  CREATECONTEXT( &gadptr )

<>  CREATEGADGET(kind,previous,newgad,tag1,tag1data,...)

     Note: CREATEGADGET looks at the 'tag' to determine how it will
           interpret the 'tagdata'. A table of 'special' tags is used

                                   APIG expects
             TAG                   TAGDATA to be
           GTTX_Text            -- stringvar/literal
           GT_Underscore        -- stringvar/literal
           GTSL_LevelFormat     -- stringvar/literal
           GTST_String          -- stringvar/literal
           GA_Text              -- stringvar/literal
           GTCY_Labels          -- pointer
           GTMX_Labels          -- pointer
           GTLV_Labels          -- pointer
           GTLV_ShowSelected    -- pointer
           GTMN_TextAttr        -- pointer
           GT_VisualInfo        -- pointer
           GTMN_Menu            -- pointer
           GTMN_SecondaryError  -- pointer
           all others           -- numeric

           What does all this mean ?
           eg. from RKM autodocs GTTX_Text specifies a POINTER to a
               null terminated string.

               You would think then to do something like:

               myptr = ALLOCMEM(30,MEMF_CLEAR)
               call export(myptr,"MY TEXT")
               CREATEGADGET(... ,GTTX_Text,myptr, ... )

               Nope, dont do that.

               APIG sees GTTX_TEXT and expects tagdata to be strvar/literal
```

```
                Thus simply do

                CREATEGADGET(... ,GTTX_Text,"MY TEXT", ... )

                or

                mytext = "MY TEXT"
                CREATEGADGET(... ,GTTX_Text,mytext, ... )

                In this case APIG will allocate memory for "MY TEXT".
                The memory will be owned by the NewGadget struct.


<>   CREATEGADGETA(kind,previous,newgad,taglist)


<>   CREATEMENUS(newmenu,tag1,tag1data,...)

      NOTE: CREATEMENUS examines the tags the same as CREATEGADGET().


<>   CREATEMENUSA(newmenu,taglist)

     Note CREATEMENUS/A will use either APIG's internal new menu data pointer
          or one you have (somehow) hardcoded.


<>   DRAWBEVELBOX(rp,l,t,w,h,vi,GTBB_Recessed)

<>   FREEGADGETS(glist)

<>   FREEMENUS(menu)

<>   FREEVISUALINFO(vi)


<>   GETVISUALINFO(scr,tag1,tag1data,...)

      NOTE: GETVISUALINFO examines the tags the same as CREATEGADGET().


<>   GETVISUALINFOA(scr,taglist)

<>   GT_BEINGREFRESH(window)

<>   GT_ENDREFRESH(window,TRUE/FALSE)

<>   GT_REFRESHWINDOW(window)


<>   GT_SETGADGETATTRS(gad,window,requester,tags)

      NOTE: GT_SETGADGETATTRS examines the tags the same as CREATEGADGET().


<>   GT_SETGADGETATTRSA(gad,window,requester,taglist)
<>   LAYOUTMENUITEMS(menuitem,vi,tag1,tag1data,...)

      NOTE: LAYOUTMENUITEMS examines the tags the same as CREATEGADGET().


<>   LAYOUTMENUITEMSA(menuitem,vi,taglist)


<>   LAYOUTMENUS(menu,vi,tag1,tag1data,...)

      NOTE: LAYOUTMENUS examines the tags the same as CREATEGADGET().


<>   LAYOUTMENUSA(menu,vi,taglist)
```

1.92 Utility library functions

```
                         Utility Library Functions

 After parameter conversion the following functions result in a
 direct call to the Amiga Utility Library function of the same name.
 Please refer to your favorite Amiga reference book(s) for a
 detailed description.


<>   ALLOCATETAGITEMS(N)

<>   AMIGA2DATE(amigatime,date)

<>   CHECKDATE(date)

<>   CLONETAGITEMS(taglist)

<>   DATE2AMIGA(date)

<>   FILTERTAGCHANGES(changelist,oldvalues,apply)

<>   FILTERTAGITEMS(taglist,tagarray,logic)

<>   FINDTAGITEM(tagval,taglist)

<>   FREETAGITEMS(taglist)

<>   GETTAGDATA(tagval,default,taglist)

<>   MAPTAGS(taglist,maplist,includemiss)

<>   NEXTTAGITEM(tagitemptr)

<>   PACKBOOLTAGS(intialflags,taglist,boolmap)

<>   REFRESHTAGITEMCLONES(clonetagitems,originaltagitems)
<>  TAGINARRAY(tag,tagarray)
```

1.93 Menu Hints

```
 Some Menu Hints


    Menu Text

          You will find that using pre-constructed IntuiText allows for
          better placement of the menu/item/subitem text.  When menu text
          is specified in the make... call APIG builds a simple IntuiText
          structure.

          Using pre-constructed IntuiText, you could make the select
          box as large as you like and place the text anywhere within
          the select box (since the IntuiText will be relative to its
          containing menu/item/subitem select box.

          eg.

            instead of this

            (1)    myitem = makeitem(w,"My Item Text",...,0,0)

            do this

                   myitemIntuiText = makeitext(w,'My Item Text',...)
            (2)    myitem = makeitem(w,"",...,0,myitemIntuiText)

                   (dont forget that the IntuiText can be linked too!)
                   (allowing multiple text be displayed for a single menu)

            Form (1) allows you to quickly get the menu structure/layout
            built, afterwhich you can go back and fill in the details with
            pre-constructed IntuiText, form (2).


    The 'top' parameter

            It is the value of 'top' that allows you to
            position items/subitems on the same line.

            eg.
               makeitem(w,'AA',m,0,0,... )
               makeitem(w,'AB',m,0,0,... )
               makeitem(w,'AC',m,x,65536,... )
               makeitem(w,'AD',m,x,0,... )

               would produce

             +-------------------------+
             |  AA item       AC item |
             |  AB item       AD item |
             +-------------------------+
             be measured from zero. The item AD is measured
             from the item which precedes it, which is item AC,
             AD has a 'top' value of 0, therefore it is
             measured from the bottom of AC.  If AD had a
             non-zero 'top' value < 65536 then the menu would
             look something like this


             +------------------------+
             |  AA item       AC item |
             |  AB item               | <---
             |                        |  AD top pixels (ADtop < 65536)
             |                        |
             |                AD item | <---
             +------------------------+

             there are 'ADtop' number of pixels between
             the BOTTOM of AC and the top of item AD.


             if item 'ADtop' value were negative then their
             would be 'ADtop' number of pixels between
             the TOP of AC and the top of item AD.

             +------------------------+ <---
             |  AA item       AC item |
             |  AB item               |
             |                        |  AD top pixels
             |                        |
             |                AD item | <---
             +------------------------+


             Note you must shift the items over 'x'
             number of pixels so that AC does overlay
             item AA.



             If item AC had a 'top' value > 65536 then the
             menu would look something like this

             +------------------------+ <---
             |  AA item               |     mod(ACtop,65536)
             |  AB item               |     pixels
             |                AC item | <---
             |                        |
             |                        |
             |                        |
             |                AD item |
             +------------------------+


             item AD will still be measured relative to the one
             that precedes, ie. item AC.
```

depending on the 'top' values for items AC and AD.

```
          +--------------------------------+
          |  AA item   AC item    AD item  |
          |  AB item                       |
          |                                |
          +--------------------------------+

              or this

          +--------------------------------+
          |  AA item   AC item             |
          |  AB item                       |
          |                                |
          |                       AD item  |
          +--------------------------------+
```

```
    The 'height' parameter

            The 'height' parameter determines how tall the select box is,
            for sub-items you will want the select boxes to be close
            together, so that the menu layer is not re-drawn as you move
            from sub-item to sub-item.
            If your sub-items flicker/jump/wiggle or whatever you call it,
            then you probably will want to adjust the height parameter.
```

1.94 APIG pointers

```
A little about APIG pointers.

In version 0.5 APIG simply returned a pointer (ACTUAL address) to
the allocated structure, APIG knew nothing about the pointer or
what it pointed to.  In fact one could simply hand hardcode a
block of memory for a particular structure and APIG 0.5 would
use it.

Versions greater than 0.5 verify that the pointer you passed in a
function call, points to a structure of the appropriate type.
It does this by storing information about the pointer at negative
offsets, similar to ARexx argstrings.

ALL pointers returned (made) by APIG have the following information
stored at negative offsets from the pointer:

format 1: APIGTAG

   contents | $4AFA | ptr | type | structure memory block |
   offset   |  -8   | -6  |  -2  |
   size     |   2   |  4  |   2  |
                                 ^ APIG gives you this addr

   $4AFA - 2bytes Motorola says this will be illegal/trap forever
   ptr   - 4bytes address of structure (points back to the structure)
   type  - 2bytes numeric type of data structure allocated by APIG
```

```
APIG will check the validity of all pointers passed to it and return
(fail) if it is not formatted as above.


If the allocated structure is a BitMap or RastPort allocated with
MAKEBITMAP/MAKERASTPORT, or a bitmap/image loaded with LOADIFF or
LOADIMAGE then the following format is used.

format 2: APIGTAG

   contents | bmhdr | $4AFA | ptr | type | structure memory block |
   offset   |  -12  |  -8   | -6  |  -2  |
   size     |   4   |   2   |  4  |   2  |
                                         ^ APIG gives you this addr

   bmhdr - 4bytes, points to the IFF BMHD header that was loaded
           with the IFF bitmap/image by iff.library.  If the
           APIG IFF functions do not provide all the information
           you need, you can use GETVALUE() with proper offsets to
           get the information from the BMHD itself.
           (See bmhdr format below)

   The IFF functions, eg. USEIFFCOLOR, do check the validity of
   the 'bmhdr' pointer.


   If MAKEBITMAP/MAKERASTPORT were used to create the pointer then
   the 'bmhdr' pointer will be NULL.

   Any function that uses a bitmap/rastport pointer, will NOT check
   for validity, since you can grab a bitmap/rastport from any
   window/screen/layer.


The following are the current values used for 'type' at offset -2,
use these values with the MAKESTRUCT/MAKEPOINTER functions.

Type

 00  = just a block of memory
 15  = Gadget (bool/string/prop)
 21  = IntuiText
 27  = Requester
 29  = EasyStruct
 33  = Border
 39  = Bitmap
 45  = Rastport
 51  = Image
 57  = Menu
 63  = MenuItem
 69  = SubItem
 75  = PropInfo   (made internally, for gadgets)
 81  = StringInfo (made internally, for gadgets)
 87  = Text Attr
 95  = NewGadget
101  = NewMenu
107  = TextExtent
```

```
     400  = apignmdata

Note: Window/Screen pointers are the pointers returned by Intuition
      OpenWindow/OpenScreen functions, and do NOT have the above format.
      Also GadTool/AslRequester pointers do NOT have the above format
      either, GadTool/AslRequester pointers are the actual pointers
      returned by the Intuition function.  In general if it is a pointer
      that I 'MAKE' then it has the above format, if it is a pointer that
      results from an Intuition call I dont touch it. (eg. CREATEMENU())



  Given the above, a new function has been added to allow hard coding of
  structures.


<>  MAKESTRUCT(owner,type,size,memtype) <>  MAKEPOINTER - synonymous with  ↩
   MAKESTRUCT

      Allocate a null structure/memory block.

      Inputs:

          owner      - pointer to object which will own this structure.
                       See description of MAKEBOOLGAGET.

          type       - numeric, type as specified above
                       The size allocated is implicit in the type
                       you specify.

          size       - numeric, size of mem block to allocate
                       APIG uses 'type' to determine the amount
                       of memory to allocate. If 'size' is non-zero
                       then it specifies an extra number of bytes
                       to be allocated in addition to the standard
                       structure size.

          memtype    - 4byte hex-string, memory type to allocate
                       (MEMF_CHIP/MEMF_CLEAR/MEMF_FAST/MEMF_PUBLIC)

     Returns:        - pointer to allocated structure as a rexx hex string.
                     - returns null ('0000 0000'x) if call fails


     eg.  allocate memory for a gadget structure

          mygad = allocstruct(0,15,0,MEMF_CLEAR)

          returns an APIG pointer, formatted as above with negative offset
          values, to a null memory block the size of a Gadget structure.

          The 'owner' is 0, indicating an independent structure which must
          be explicitly freed (using FREETHIS()).  You could make 'owner'
          a window, so that when the window is closed the memory is
          freed.

          Note the 'size' parm is 0, type 15 tells APIG how much memory
```

```
        to allocate.  If 'size' were non-zero then APIG would allocate
        the standard structure size + 'size' bytes.

        ie.  mygad = allocstruct(0,15,30,MEMF_CLEAR) would allocate
             an additional 30 bytes, along with the 'standard' size of
             a Gadget structure.


    eg.  allocate a block of memory

        mymem = allocstruct(0,0,400,MEMF_CLEAR)
        or
        mymem = allocpointer(0,0,400,MEMF_CLEAR)

        'owner' is zero, so must be freed explicitly
        'type'  is zero, thus ...
        'size'  determines total amount of memory to actually allocate

  After allocating the structure SETVALUE can then be used to assign
  values to the structures data fields.



  If you were hard coding an image structure you might do the
  following:


  image = makestruct(0,51,0,MEMF_CLEAR)


  imagedata = makepointer(image,0,72,MEMF_CHIP)

  (note imagedata is just a block of mem, 'owned' by 'image',
        thus when 'image' is freed so is the imagedata block)


  x = setvalue(image,0,2,'n',0,0)            /* left edge   */
  x = setvalue(image,2,2,'n',0,0)            /* top edge    */
  x = setvalue(image,4,2,'n',16,0)           /* width       */
  x = setvalue(image,6,2,'n',9,0)            /* height      */
  x = setvalue(image,8,2,'n',4,0)            /* depth       */
  x = setvalue(image,10,4,'p',imagedata,0)   /* imagedata   */
  x = setvalue(image,14,1,'n',255,0)         /* planepick   */
  x = setvalue(image,15,1,'n',255,0)         /* planeonoff  */
  x = setvalue(image,16,4,'p',0,0)           /* next image  */

  Assuming, of course you have stuffed some data into imagedata,
  the 'image' pointer can now be used with the DrawImage() function.

  If the ALLOCMEM() function had been used to allocate the block
  then used in DrawImage(), DrawImage would have found the pointer
  to be invalid and simply would have returned (failed).



 The 'bmhdr' pointer points to the following structure:
STRUCTURE   iffinfoblock,0

     UWORD  iffviewmode              ; offset 0, View Modes for the IFF

     UWORD  iffcolors                ; offset +2, number of colors

     STRUCT bmheader,SIZEOF_bmhd
                                     ; BitMapHeader Info from the IFF file
                                     ; as shown below

     STRUCT cmheader,128             ; offset +24
                                     ; ColorMap Info from the IFF file
                                     ; max 64 colors @ 2bytes per color


-------------------------

STRUCTURE BitMapHeader,0
     UWORD bmh_Width                 ; offset +4 from bmhdr pointer
     UWORD bmh_Height                ; offset +6
     WORD  bmh_XPos                  ; offset +8
     WORD  bmh_YPos                  ; offset +10
     UBYTE bmh_nPlanes               ; offset +12
     UBYTE bmh_Masking               ; offset +13
     UBYTE bmh_Compression           ; offset +14
     UBYTE bmh_Pad1                  ; offset +15
     UWORD bmh_TranspCol             ; offset +16
     UBYTE bmh_XAspect               ; offset +18
     UBYTE bmh_YAspect               ; offset +19
     WORD  bmh_PageWidth             ; offset +20
     WORD  bmh_PageHeight            ; offset +22


eg. what APIG effectively does to obtain the width of an IFF pic.

    bmhdr = getvalue(pic,-12,4,'p')
    width = getvalue(bmhdr,4,2,'n')
```

1.95 Sorted function reference

```
                 This node is sorted by the alphabet.

 -----------


   @{er)" link " link "ActivataGadID" 0}
" link    @{id,w
               indow,requester)
               " link "ADD_LIST_NODE" 0}
   @{w(window)nctions" link "Intuition library functions" 17}
   @{itio
               n,~nodesize,~pri,~
               type)NODE
               " 0}
```

```
                @{rary functions" link "ASL library functions" 12}
@{List
             (window,gadget,posi
             tion
             ,numgad,requester)
             " link ry functions" link "Utility library functions" 12}
@{d(list,node)y functions" link "Graphics Library functions" 15}


             A@{Exec
              library fu
             nctions" link "Graphics Library functions" 21}
@{ddTo
             _NewMenu(apignmdata,type,l
             abel
             ,commkey,flags,mutual,usrdata)
             " link "Intuition library functions" 19}
@{ocASLRequest(type,tags) link ctions" link "Graphics Library functions" 23}
@{Item
             s(N)
             " link "Graphics Library functions" 25}
@{ "ASL library functions" link "Graphics Library functions" 27}
@{y functions" link "Graphics Library functions" 29}
@{atim
             e,date)
             " link "Graphics Library functions" 31}
@{link " link "Graphics Library functions" 34}

                @{
             "Are
             ADraw(rp,x,y)
             " link "Graphics Library functions" 36}
@{eaEl
             lipse(rp,cx
             ,cy,a,b)Grap
             hics Librar
             y functions" link "Intuition library functions" 27}
@{rary functions" link "Intuition library functions" 29}
@{"Gra
             phics Library f
             unctions" link "Intuition library functions" 31}
@{uest
             (req,tags,owner)
             " link ons"
              25}
                @{st(w
             indow,itext,itext,i
             text,posflags,newflags,width,hgt)

             functions
             " 19
             }
                @{indow)tion library functions" link "Graphics Library ←
                    functions" 57}


              @{ link unctions" link "CONVERTRAWKEY" 0}
```

```
   @{s)" link " link "layers library functions" 16}
   @{srcb
                m,srcx,srcy,dstbm,dstx,
                dsty,sizex,sizey,minterm,mask,tempa)

                Lib
                rary functions" link "Gadtools library functions" 62}
   @{bm,s
                rcx,srcy,rp,destx,desty,sizex,sizey,mi
                nterm)

                fu
                nctions" link "layers library functions" 24}
   @{ "Gr
                aphics Library f
                unctions" link "layers library functions" 32}
   @{tMas
                kBitmapRastPort(scr
                bm,s
                rcx,srcy,rp,destx,des
                ty,sizex,sizey,minterm,bltmask)

                "Graphics Library function
                s" 3
                1}
                   @{Pattern(rp,mask,x1,y1,x2,y2,bytecnt)raphics Library functions ←
                      " link "Intuition library functions" 59}
   @{te(s
                rctemplate,srcx,srcmod
                ,rp,dstx,dsty,sizex,sizey)ink "Graphics Library functions

                36}
                   @{MDep
                th(bm)
                " link "Intuition library functions" 63}
   @{" link "Intuition library functions" 63}

                   @
                {"BM
                Width(bm)
                " link "Intuition library functions" 65}
   @{struct,idcmp,args)tions" link "APIG list related functions" 9}
   @{ange
                WindowBox(window,left,top,w
                idth,height)brary functions" link "Exec library functions" 22}


                  tions" link "Utility library functions" 22}
   @{ink
                "Intuition library
                functions" link "Utility library functions" 24}
   @{p,x,y)ibrary functions" link "Graphics Library functions" 67}
" link    @{ink "Intuition library functions" link "FREE_EXEC_LIST" 0}
   @{ndow)ons"
                 43}
                   @{Screen(rp,x,y)Library functions" link "FREE" 0}
                  estr
                p,destx,desty,xsize,ysiz
                e,minterm)hics
                 Library
                functions" link "FREE" 0}


                  nk "
                Utility library
                 functions" link "FREE" 0}
   @{t)" link ctions" link "Intuition library functions" 126}
   @{itor(monitor_spec)"Graphics Library functions" link "FREE" 0}
   @{(screen)functions" link "Exec library functions" 24}

                   @
                {"CloseWindow(window) link unctions" link "GETARRAY" 0}
   @{y(keycode,qualifier,keymap)NVERTRAWKEY" link "Intuition library functions"  ←
      132}
   @{ayer
                (windowpointer,x0,y0,x1,y1,f
                lags
                ,bm2)
                " link "Graphics Library functions" 71}
   @{dtoo
                ls library funct
                ions
                " 11}
                   @{adge
                t(kind,previous,newgad,
                tag1,tag1data,...)tool
                s library funct
                ions
                " 13}
                   @{tA(kind,previous,newgad,taglist)dtools library functions"  ←
                      link "Intuition library functions" 138}
   @{g1,tag1data,...) functions" link "Utility library functions" 28}
   @{menu,taglist)ary functions" link "Gadtools library functions" 85}
   @{yer(
                windowpointer,x0,y0,x1,y1,f
                lags,bm2)ibrary functions" link "GETWINDOWLAYER" 0}

                   @
                {"Date2Amiga(date)lity library functions8}
   @{r)" link nctions" link "Gadtools library functions" 94}
   @{n)" link functions" link "Gadtools library functions" 99}
   @{ "Intuition library functions" link "Gadtools library functions" 104}
   @{(sta
                rtsecs,startmicros,curre
                ntsecs,currentmicros) "In
                tuition library fu
                nctions

                57}
                   @{rp,x,y)cs Library functions59}
   @{p,l,
                t,w,h,vi,GTBB_Rec
                essed)ols
                library function
                s" 7
                6}
                   @{rder
                (rp,bord
                er,left
                offset,top
                offset)" link ition l
                ibrary functions
                " 59
                }
                   @{" link unctions" link "Intuition library functions" 140}
   @{,cy,a,b)s Library functions}
   @{link " link "Gadtools library functions" 112}


                 @{offset,state,drinfo)tion
                 library functi
                ons"

                63}
                   @{wind
                ow,title,bodytext,gadtext,arglist,IDCMP,fla
                gs)" link

                functi
                ons"
                 65}
                   @{ndow,es,IDCMP_ptr,ArgList)uition library functions" link " ↩
                      MAKEAREA" 0}
   @{ist(list)ions
                " 9}
                   @{" link "MAKEBOOLGADGET" 0}
   @{t(re
                quester,window)
                " link "MAKEBORDER" 0}


                E@{" link "MAKEITEM" 0}
   @{ues,
                apply)
                " link "MAKEITEXT" 0}
   @{ms(t
                aglist,tagarray,logic)
                " link "MAKEMENU" 0}
" link    @{" link "MAKENEWGADGET" 0}
   @{ndTa
                gItem(tagval,t
                aglist)ility library functions" link "MAKEPROPGADGET" 0}
   @{" link "MAKEPROPGADGET" 0}
   @{ode,
                x,y)
                " 67
                }
                   @{" link "MAKEREQUESTER" 0}
   @{(lis
                tptr,~nodestructsize,~liststructsize)
                " link "MAKESTRGADGET" 0}
   @{"FREE_EXEC_NODE" link "MAKE" 0}
   @{w)" link " link "MAKE" 0}
   @{uest
                (filerequest)
                " link "MAKESUBITEM" 0}
   @{" link "MAKESUBITEM" 0}

                   @{
                "FreeBitmap(pointertobitmap)E" 0
                }
                   @{nk "
                ASL library functio
                ns"
                97}
                   @{tools library functions 78}
   @{e)" link " link "Intuition library functions" 156}
   @{}
   @{k "G
                adtools libr
                ary functions" link "layers library functions" 40}
   @{reeR
                astPort(pointertorastport)
                " link "layers library functions" 37}
" link    @{nk "Intuition library functions}
   @{ion
                library funct
                ions
                " 126}
                   @{" link "Intuition library functions" 158}
" link    @{dent~structure)FREE
                " 0}
                   @{ppoi
                nter)
                " link "Utility library functions" 32}


                  ry functions" link "Intuition library functions" 176}
   @{sualInfo(vi)functions" link "Intuition library functions" 180}
   @{ted(gadgetptr)GADSELECTED" link "Graphics Library functions" 92}
   @{arra
                yindx)
                " link "OPENSCREEN" 0}
   @{ link " link "Intuition library functions" 182}
   @{" link "OPENWINDOW" 0}
   @{ffer
                ,size)
                " link "Intuition library functions" 185}

                   @{
                "GetDisplayInofData(handle,buf,size,tag,id)link " link "PITEXT" 0}

                   @
                {"Ge
                tGadPTR(window,gadgetid,
                requ
```

```
             ester)
             " link "Intuition library functions" 197}
@{P(wi
             ndow)
             " link "Intuition library functions" 199}
@{LayerInfo(layer)k "GETLAYERINFO" link "Graphics Library functions" 99}
@{stPo
             rt(layer)
             " link "Intuition library functions" 201}


              @{brary functions" link "Utility library functions" 36}
@{BITMAP" link "Intuition library functions" 205}
@{ap(screen)"GET
             SCREENBITMAP
             " 0}
@{,size,type,screen)ibrary functions 136
             }
                @{creen)ry functions" link "ASL library functions" 99}
@{" link "ASL library functions" 99}

                @
             {"Ge
             tSTRGad(window,gadgetid,req
             uest
             er)
             " link "SAVEIFF" 0}
@{al,d
             efault,taglist)
             " link "SAVEIFFCLIP" 0}

                @
             {"GetValue(ptr,offset,size,type)VALUE" link "Intuition library  ←
                functions" 273}
@{lInf
             o(scr,tag1,tag1data,.
             ..)" link functions" link "Graphics Library functions" 103}
@{t)" link ary functions" link "SETARRAY" 0}


             G@{phic
             s Library funct
             ions
             " 73}
                @{link R" 0
             }
                @{Port(window)T" 0
             }
                @{aypt
             r,xindex)
             0}
@{tr,y
             index)
             " link "SETGADTYPE" 0}
@{GT_BeginRefresh(window)ibrary functions" link "Intuition library functions"  ←
    281}
```

```
@{T_EndRefresh(window,TRUE/FALSE)nk "Gadtools library functions" link " ←↩
   SETNEWGADGET" 0}
@{tool
             s library funct
             ions
             " 96}
                @{)" link  99}
@{A(ga
             d,window,reques
             ter,taglist) functions" link "SETSELECT" 0}
@{dy(p
             ropgadgetptr)
             }
@{r)" link " link "SETSTRGADID" 0}
@{" link " link "SETTAGSLOT" 0}
@{)" link " link "SETVALUE" 0}
@{ter)" link " link "Intuition library functions" 301}
@{ "IFF" link "Graphics Library functions" 129}
@{ode(
             pointer)
             " 0}
@{ink "IFF" link "Intuition library functions" 303}
@{imag
             e)
             " link "layers library functions" 50}

                @
             {"IM
             GHeight(image)
             " link "Intuition library functions" 305}


              @{ink "IMG" link "Intuition library functions" 309}
@{ry functions ()" link "Utility library functions" 38}
@{nitBitmap Library functions ()" link "Graphics Library functions" 138}
@{ort" link  ()" link "Graphics Library functions" 140}
@{ntuition library functions}
@{e,li
             stnode)
             " link "Intuition library functions" 311}
@{itext)ary functions" link "VERT" 0}
@{temA
             ddress(menustrip,menu
             number)tuition library functions140}
@{(menunumber)"Int
             uition library function
             s" 1
             42}
                @{ctions" link "Intuition library functions" 323}
@{Menu
             Items(menuitem,vi,tag
             1,tag1data,...)Gadtools library functions07}
@{emsA
             (menuitem,vi,tagl
             ist)" link "Gadtools library functions" 112}

             LayoutMenus(menu,vi,tag1,tag1data,...)
```

```
                LayoutMenusA(menu,vi,taglist)

                ListEmpty(list)

                LoadIFF(filename,owner)

                LoadImage(filename,imageptr,left,top,owner)

                LoadRGB4(screen,arrayptr,count)

                LockPubScreen(screename/null)

                LockPubScreenList(screename/null)

                MakeArea(window,xsize,ysize,maxvectors)

                MakeBitmap(width,height,depth,owner)
                   MakeBoolGadget(owner,left,top,width,hgt,flags,activation,itext, ←
                      bpen,render,select,gadid,linkto)
link "MAKEBOOLGADGET" 0}

                MakeBorder(owner,arrayptr,arraycnt,left,top,fp,bp,dm,linkto)
                   MakeItem(menustrip,text,menu,left,top,width,height,flags,ME,COM ←
                      ,fp,bp,dm,itemfill,selectfill)
link "MAKEITEM" 0}

                MakeIText(owner,text,xpos,ypos,fpen,bpen,dmode,fontattr,linkto)

                MakeMenu(menuowner,menutext,leftedge,width,flags,menupointer)

                MakeNewGadget(vinfo,font,left,top,width,height,text,flags,id, ←
                   usrdata)

                MakeNewMenu(n)

                MakePointer()
                   MakePropGadget(owner,left,top,width,hgt,flags,activation,itext ←
                      ,,piflags,hbody,vbody,gadid,linkto
,knobimage)" link "MAKEPROPGADGET" 0}

                MakeRastPort(width,height,depth,owner)
                   MakeRequester(window,left,top,width,height,gadget,text,border, ←
                      backfill,flags,relleft,reltop,bm)
link "MAKEREQUESTER" 0}
   MakeStrGadget(window,left,top,width,hgt,flags,activation,itext,bpen,render, ←
      select,gadid,linkto,s
trlen,undobuf)" link "MAKESTRGADGET" 0}

                MakeStruct(owner,type,size,mem_type)

                MakeStruct(owner,type,size,memtype)
                   MakeSubItem(menustrip,text,item,left,top,width,height,flags,ME, ←
                      COM,fp,bp,dm,itemfill,selectfill)
" link "MAKESUBITEM" 0}

                MakeTAttr(window,fontname,fontsize)
```

```
                 MapTags(taglist,maplist,includemiss)

                 MenuNum(menunumber)

                 MenuNumber(menustrip,menu,item,subitem)

                 ModeNotAvailable(id)

                 ModifyIDCMP(window,idcmpflags)
                    ModifyProp(gadget,window,requester,flags,hpot,vpot,hbody,vbody)
functions" 156}

                 MouseFrequency(window,N)

                 Move(rp,x,y)

                 MoveLayer(layer,dx,dy)

                 MoveLayerInFrontOf(layertomove,targetlayer)

                 MoveScreen(screen,deltax,deltay)

                 MoveWindow(window,deltax,deltay)

                 MoveWindowInFrontOf(window,behindwindow)

                 NewList(list)
                    NewModifyProp(gadget,window,requester,flags,hpot,vpot,hbody, ←
                       vbody,numgad)
library functions" 158}

                 NewObjectA(class,classid,taglist)

                 NextDisplayInfo(id)

                 NextPubScreen(screen,namebuffer)

                 NextTagItem(tagitemptr)

                 OffGadget(gadget,window,requester)

                 OffMenu(window,menunumber)

                 OnGadget(gadget,window,requester)

                 OnMenu(window,menunumber)

                 OpenFont(textAttr)

                 OpenMonitor(monitor_name,display_id)

                 OpenScreen(left,top,width,height,depth,dpen,bpen,vmodes,type,title ←
                    )

                 OpenScreenTagList(newscreen,taglist)
```

```
                   OpenWindow(portname,left,top,wid,hgt,dpen,bpen,IDCMP,flags, ↩
                      title,scr,console,bitmap,chkmark,gadl
ist)" link "OPENWINDOW" 0}

                OpenWindowTagList(portname,newwindow,taglist,console)

                PackBoolTags(intialflags,taglist,boolmap)

                PIText(rp,left,top,text,fp,bp,dm,font)

                PolyDraw(rp,count,array)

                PrintIText(rp,itext,leftoffset,topoffset)

                PubScreenStatus(screen,statusflags)

                ReadPixel(rp,x,y)

                RectFill(rp,xmin,ymin,xmax,ymax)

                RefreshGadgets(gadgets,window,requester)

                RefreshGList(gadgets,window,requester,numgad)

                RefreshTagItemClones(clonetagitems,originaltagitems)

                RefreshWindowFrame(window)

                RemHead(list)

                Remove(node)

                RemoveGadget(window,gadget)

                RemoveGList(window,gadget,numgad)

                RemTail(list)

                ReportMouse(boolean,window)

                Request(requester,window)
                   RequestFile(req,multi,save,hail,dir,file,pat,nofile,win,left, ↩
                      top,width,hgt,sep)
library functions" 99}

                ResetMenuStrip(window,menu)

                SaveIFF(bitmap,filename,colortab,HAM,compress)

                SaveIFFClip(bitmap,filename,x,y,w,h,colortab,HAM,compress)

                ScaleRDiv(factor,numerator,denominator)

                ScreenToBack(screen)

                ScreenToFront(screen)
```

```
ScrollLayer(layer,dx,dy)

ScrollRaster(rp,dx,dy,xmin,ymin,xmax,ymax)

ScrProcName(scrptr)

Set_Apig_Globals()
   RatAFAl&4(0a

PubScreenStatus(screen,statusflags)

ReadPixel(rp,x,y)

RectFill(rp,xmin,ymin,xmax,ymax)

RefreshGadgets(gadgets,window,requester)

RefreshGList(gadgets,window,requester,numgad)

RefreshTagItemClones(clonetagitems,originaltagitems)

RefreshWindowFrame(window)

RemHead(list)

Remove(node)

RemoveGadget(window,gadget)

RemoveGList(window,gadget,numgad)

RemTail(list)

ReportMouse(boolean,window)

Request(rtPrefs(prefbuffer,size,inform)

SetPubScreenModes(modes)

SetRast(rp,pen)

SetRGB4(screen/window,pen,r,g,b)

SetSelect(gadgetptr,state)

SetSoftStyle(rp,style,enable)

SetStrGad(gadgetptr,text)

SetStrGadID(window,gadid,text,requester)

SetTagSlot(tagarray,slot,tag,'p'/'n',value)

SetValue(ptr,offset,size,type,value,len)

SetWindowTitle(window,widnowtitle,screentitle)
```

```
                SetWRMSK(rp,wrtmask)

                SetX(arrayptr,xindex,value)

                SetY(arrayptr,yindex,value)

                ShowTitle(screen,showit)

                SizeLayer(layer,dx,dy)

                SizeWindow(window,deltax,deltay)

                SubNum(menunumber)

                SysReqHandler(window,idcmpflagsptr,waitinput)

                TagInArray(tag,tagarray)

                Text(rp,string,count)

                TextExtent(rp,string,count,textextent)
                   TextFit(rp,string,len,textextent,consextent,strdir,consbitwid, ↩
                      consbithgt)
Library functions" 140}

                TextLength(rp,string,count)

                TickFrequency(window,N)

                UnlockPubScreen(screenname/null,screenptr)

                UpFrontLayer(layer)

                UseIFFColor(pointer,scr)

                VertBody(propgadgetptr)

                VertPot(propgadgeptr)

                ViewAddress()

                ViewPortAddress(window)

                WbenchToBack()

                WbenchToFront()

                WindowInfo(window,code)

                WindowLimits(window,minwidth,minheight,maxwidth,maxheight)
                21}

                WindowToBack(window)

                WindowToFront(window)

                WinTaskName(window)
```

```
WriteConsole(window,text)

WritePixel(rp,x,y)

ZipWindow(window)
```

1.96 Sorted Function Reference

```
 This node is sorted by the alphabet.
```

```
-----------
```

```
ActivateGadget(gadget,window,requester)

ActivateGadID(gadgetid,window,requester)

ActivateWindow(window)

Add_List_Node(listptr,~string,~position,~nodesize,~pri,~type)

AddGadget(window,gadget,position)

AddGList(window,gadget,position,numgad,requester)

AddHead(list,node)

AddTail(list,node)

AddTo_NewMenu(apignmdata,type,label,commkey,flags,mutual,usrdata)

AllocASLRequest(type,tags)

AllocateTagItems(N)

AllocFileRequest()

AllocVec(size,type)

Amiga2Date(amigatime,date)

AreaCircle(rp,cx,cy,radius)

AreADraw(rp,x,y)

AreaEllipse(rp,cx,cy,a,b)

AreaEnd(rp)

AreaMove(rp,x,y)

ASLRequest(req,tags,owner)
```

```
                  AutoRequest(window,itext,itext,itext,posflags,newflags,width, ↩
                     hgt)
functions" 19}

               BeginRefresh(window)

               BehindLayer(layer)

               BitmapScale(bitscaleargs)
                  BltBitmap(srcbm,srcx,srcy,dstbm,dstx,dsty,sizex,sizey,minterm, ↩
                     mask,tempa)
Library functions" 25}
   BltBitmapRastPort(srcbm,srcx,srcy,rp,destx,desty,sizex,sizey,minterm)
functions" 27}

               BltClear(memblock,bytecount,flags)
                  BltMaskBitmapRastPort(scrbm,srcx,srcy,rp,destx,desty,sizex, ↩
                     sizey=((j
?????
???
??J
??J
?IÝZ
JÉ]ÐH?ZHÈR
HØH X
È
J??uT??+uuuPz#RX?  0(}rx)cux(B*auxp Ý\?)UW}PPXPÒ
B?R?PX¡éUz?HA}RÒ  U|8?U}}zø?RÒ
H((]XP  IUx?H?A}RÒ  U|8?U}}zø?RÐ(H?*¨h?
((((?*è???X?H?(???(h???(?(@-?H(@?=?h???IUx?H?A}RH?((H(H??x?HP?h?  x?(?@(H???U|???U ↩
   }}zø?RÀXh??*H?(??Ð??X?H?(????(?-?(?=??H((]?X(H?X-IUx?H?A}RH?((H(H??"Exec  ↩
   library functions" 18}

               AddTo_NewMenu(apignmdata,type,label,commkey,flags,mutual,usrdata)

               AllocASLRequest(type,tags)

               AllocateTagItems(N)

               AllocFileRequest()

               AllocVec(size,type)

               Amiga2Date(amigatime,date)
                  AreaCircle(rp,cx,cy,radius)

               CloneTagItems(taglist)

               CloseFont(font)

               CloseMonitor(monitor_spec)

               CloseScreen(screen)

               CloseWindow(window)

               ConvertRawKey(keycode,qualifier,keymap)
```

```
                CreateBehindLayer(windowpointer,x0,y0,x1,y1,flags,bm2)

                CreateConText(&gadptr~)

                CreateGadget(kind,previous,newgad,tag1,tag1data,...)

                CreateGadgetA(kind,previous,newgad,taglist)

                CreateMenus(newmenu,tag1,tag1data,...)

                CreateMenusA(newmenu,taglist)

                CreateUpFrontLayer(windowpointer,x0,y0,x1,y1,flags,bm2)

                Date2Amiga(date)

                DeleteLayer(layer)

                DisplayBeep(screen)

                DisposeObject(object)

                DoubleClick(startsecs,startmicros,currentsecs,currentmicros)
                57}

                Draw(rp,x,y)

                DrawBevelBox(rp,l,t,w,h,vi,GTBB_Recessed)

                DrawBorder(rp,border,leftoffset,topoffset)

                DrawCircle(rp,cx,cy,r)

                DrawEllipse(rp,cx,cy,a,b)

                DrawImage(rp,image,leftoffset,topoffset)

                DrawImageState(rp,image,leftoffset,topoffset,state,drinfo)
                63}
   EasyRequest(window,title,bodytext,gadtext,arglist,IDCMP,flags)
functions" 65}

                EasyRequestArgs(window,es,IDCMP_ptr,ArgList)

                EmptyList(list)

                EndRefresh(window,complete)

                EndRequest(requester,window)

                EnQueue(list,node)

                FilterTagChanges(changelist,oldvalues,apply)

                FilterTagItems(taglist,tagarray,logic)
```

```
FindDisplayInfo(id)

FindTagItem(tagval,taglist)

FirstNode(list,node)

Flood(rp,mode,x,y)

FontExtent(font,fontextent)

Free_Exec_List(listptr,~nodestructsize,~liststructsize)

Free_Exec_Node(nodeptr,~nodesize)

FreeArea(window)

FreeASLRequest(filerequest)

FreeBIRASIM(pointer)

FreeBitmap(pointertobitmap)

FreeFileReq(filerequest)

FreeGadgets(glist)

FreeImage(pointertoimage)

FreeIText(intuitextpointer)

FreeMenus(menu)

FreeRastPort(pointertorastport)

FreeScreenDrawInfo(drinfo)

FreeSysRequest(window)

FreeTagItems(taglist)

FreeThis(pointer~to~any~independent~structure)

FreeThisMenu(menustrippointer)

FreeVec(vecptr)

FreeVisualInfo(vi)

GadSelected(gadgetptr)

GetArray(arrayptr,arrayindx)

GetAttr(attrid,object,storageptr)

GetDefaultPubScreen(namebuffer)

GetDefPrefs(prefbuffer,size)
```

```
GetDisplayInofData(handle,buf,size,tag,id)

GetGadPTR(window,gadgetid,requester)

GetIDCMP(window)

GetLayerInfo(layer)

GeTLayerRastPort(layer)

GetPrefs(prefbuffer,size)

GetRPBitmap(rp)

GetScreenBitmap(screen)

GetScreenData(buffer,size,type,screen)

GetScreenDrawInfo(screen)

GetScreenRastPort(screen)

GetSTRGad(window,gadgetid,requester)

GetTagData(tagval,default,taglist)

GetValue(ptr,offset,size,type)

GetVisualInfo(scr,tag1,tag1data,...)

GetVisualInfoA(scr,taglist)

GetVPModeID(viewport)

GetWindowLayer(window)

GetWindowRastPort(window)

GetX(arrayptr,xindex)

GetY(arrayptr,yindex)

GT_BeginRefresh(window)

GT_EndRefresh(window,TRUE/FALSE)

GT_RefreshWindow(window)

GT_SetGadgetAttrs(gad,window,requester,tags)

GT_SetGadgetAttrsA(gad,window,requester,taglist)

HorizBody(propgadgetptr)

HorizPot(propgadgetptr)
```

```
IFFColors(pointer)

IFFColorTAB(pointer)

IFFDepth(pointer)

IFFHeight(pointer)

IFFViewMode(pointer)

IFFWidth(pointer)

IMGDepth(image)

IMGHeight(image)

IMGWidth(image)

InitArea
 ()

InitBitmap
 ()

InitRastPort
 ()

InitRequester~()

InsertNode(list,node,listnode)

IntuiTextLength(itext)

ItemAddress(menustrip,menunumber)

ItemNum(menunumber)

LastNode(list,node)

LayoutMenuItems(menuitem,vi,tag1,tag1data,...)

LayoutMenuItemsA(menuitem,vi,taglist)

LayoutMenus(menu,vi,tag1,tag1data,...)

LayoutMenusA(menu,vi,taglist)

ListEmpty(list)

LoadIFF(filename,owner)

LoadImage(filename,imageptr,left,top,owner)

LoadRGB4(screen,arrayptr,count)

LockPubScreen(screename/null)
```

```
                LockPubScreenList(screename/null)

                MakeArea(window,xsize,ysize,maxvectors)

                MakeBitmap(width,height,depth,owner)
                   MakeBoolGadget(owner,left,top,width,hgt,flags,activation,itext, ←
                      bpen,render,select,gadid,linkto)
link "MAKEBOOLGADGET" 0}

                MakeBorder(owner,arrayptr,arraycnt,left,top,fp,bp,dm,linkto)
                   MakeItem(menustrip,text,menu,left,top,width,height,flags,ME,COM ←
                      ,fp,bp,dm,itemfill,selectfill)
link "MAKEITEM" 0}

                MakeIText(owner,text,xpos,ypos,fpen,bpen,dmode,fontattr,linkto)

                MakeMenu(menuowner,menutext,leftedge,width,flags,menupointer)

                MakeNewGadget(vinfo,font,left,top,width,height,text,flags,id, ←
                   usrdata)

                MakeNewMenu(n)

                MakePointer()
                   MakePropGadget(owner,left,top,width,hgt,flags,activation,itext ←
                      ,,piflags,hbody,vbody,gadid,linkto
,knobimage)" link "MAKEPROPGADGET" 0}

                MakeRastPort(width,height,depth,owner)
                   MakeRequester(window,left,top,width,height,gadget,text,border, ←
                      backfill,flags,relleft,reltop,bm)
link "MAKEREQUESTER" 0}
   MakeStrGadget(window,left,top,width,hgt,flags,activation,itext,bpen,render, ←
      select,gadid,linkto,s
trlen,undobuf)" link "MAKESTRGADGET" 0}

                MakeStruct(owner,type,size,mem_type)

                MakeStruct(owner,type,size,memtype)
                   MakeSubItem(menustrip,text,item,left,top,width,height,flags,ME, ←
                      COM,fp,bp,dm,itemfill,selectfill)
" link "MAKESUBITEM" 0}

                MakeTAttr(window,fontname,fontsize)

                MapTags(taglist,maplist,includemiss)

                MenuNum(menunumber)

                MenuNumber(menustrip,menu,item,subitem)

                ModeNotAvailable(id)

                ModifyIDCMP(window,idcmpflags)
                   ModifyProp(gadget,window,requester,flags,hpot,vpot,hbody,vbody)
functions" 156}
```

```
                MouseFrequency(window,N)

                Move(rp,x,y)

                MoveLayer(layer,dx,dy)

                MoveLayerInFrontOf(layertomove,targetlayer)

                MoveScreen(screen,deltax,deltay)

                MoveWindow(window,deltax,deltay)

                MoveWindowInFrontOf(window,behindwindow)

                NewList(list)
                   NewModifyProp(gadget,window,requester,flags,hpot,vpot,hbody, ↩
                      vbody,numgad)
library functions" 158}

                NewObjectA(class,classid,taglist)

                NextDisplayInfo(id)

                NextPubScreen(screen,namebuffer)

                NextTagItem(tagitemptr)

                OffGadget(gadget,window,requester)

                OffMenu(window,menunumber)

                OnGadget(gadget,window,requester)

                OnMenu(window,menunumber)

                OpenFont(textAttr)

                OpenMonitor(monitor_name,display_id)

                OpenScreen(left,top,width,height,depth,dpen,bpen,vmodes,type,title ↩
                   )

                OpenScreenTagList(newscreen,taglist)
                   OpenWindow(portname,left,top,wid,hgt,dpen,bpen,IDCMP,flags, ↩
                      title,scr,console,bitmap,chkmark,gadl
ist)" link "OPENWINDOW" 0}

                OpenWindowTagList(portname,newwindow,taglist,console)
                   PackBoolTags(intialflags,taglist,boolmap)
?t=%)#`(( &p(!z`940h (v!hfdla89';;$(#1-9.?%&???D?U&4q}_u%a?~"?/9i?v!#h3%=w;p;%ia ↩
   &#0-h2q8h*;t2V20p|h33!Hi"b%b=q2q.'d("" }u}^4 t?.2?"8* ??t,%h2 (,!`( (( " *"#b   ↩
   , * *"#rut(2 8}~: "?: 0(!|*:4,8&"t"( 2 :" (""  }w}^4 t?*6?0&?""  :?  t 2,""#`  ↩
   *8"    2"(pwv(2 8}~: "?uunctions" 158}

                NewObjectA(class,classid,taglist)

                NextDisplayInfo(id)
```

```
NextPubScreen(screen,namebuffer)

NextTagItem(tagitemptr)

OffGadget(gadget,window,requester)

OffMenu(window,menunumber)
   OnGadget(gadget,window,requester)

OnMenu(window,menunumber)

OpenFont(textAttr)

OpenMonitor(monitor_name,display_id)

OpenScreen(left,top,width,height,depth,dpen,bpen,vmodes,type,title ↩
   )

OpenScreenTagList(newscreen,taglist)
   OpenWindow(portname,left,top,wid,hgt,dpen,bpen,IDCMP,flags, ↩
      title,scr,consontuition library functions

ScrollLayer(layer,dx,dy)

ScrollRaster(rp,dx,dy,xmin,ymin,xmax,ymax)

ScrProcName(scrptr)

Set_Apig_Globals()

SetAFPT(rp,pattern,patternsize)

SetAPen(rp,pen)

SetArray(arrayptr,~arrayindx,~value)

SetBPen(rp,pen)

SetDefaultPubScreen()

SetDMRequest(window,dmrequester)

SetDRMD(rp,mode)

SetDRPT(rp,linepattern)

SetFont(rp,font)

SetGadType(gadgetptr,gadtype)

SetImage(image,left,top,ppick,ponoff)

SetMenuStrip(window,menu)

SetMouseQueue(window,newlength)
```

```
                SetNewGadget(ngad,vinfo,font,left,top,width,height,text,flags,id, ↩
                   usrdata)
                0}

                SetOpen(rp,pen)

                SetPointer(window,pointer,height,width,xoffset,yoffset)
                295}

                SetPrefs(prefbuffer,size,inform)

                SetPubScreenModes(modes)

                SetRast(rp,pen)

                SetRGB4(screen/window,pen,r,g,b)

                SetSelect(gadgetptr,state)

                SetSoftStyle(rp,style,enable)

                SetStrGad(gadgetptr,text)

                SetStrGadID(window,gadid,text,requester)

                SetTagSlot(tagarray,slot,tag,’p’/’n’,value)

                SetValue(ptr,offset,size,type,value,len)

                SetWindowTitle(window,widnowtitle,screentitle)

                SetWRMSK(rp,wrtmask)

                SetX(arrayptr,xindex,value)

                SetY(arrayptr,yindex,value)

                ShowTitle(screen,showit)

                SizeLayer(layer,dx,dy)

                SizeWindow(window,deltax,deltay)

                SubNum(menunumber)

                SysReqHandler(window,idcmpflagsptr,waitinput)

                TagInArray(tag,tagarray)

                Text(rp,string,count)

                TextExtent(rp,string,count,textextent)
                   TextFit(rp,string,len,textextent,consextent,strdir,consbitwid, ↩
                      consbithgt)
Library functions" 140}

                TextLength(rp,string,count)
```

```
TickFrequency(window,N)

UnlockPubScreen(screenname/null,screenptr)

UpFrontLayer(layer)

UseIFFColor(pointer,scr)

VertBody(propgadgetptr)

VertPot(propgadgeptr)

ViewAddress()

ViewPortAddress(window)

WbenchToBack()

WbenchToFront()

WindowInfo(window,code)

WindowLimits(window,minwidth,minheight,maxwidth,maxheight)
21}

WindowToBack(window)

WindowToFront(window)

WinTaskName(window)

WriteConsole(window,text)

WritePixel(rp,x,y)

ZipWindow(window)
```