extras

COLLABORATORS			
	TITLE : extras		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 26, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

extras

1.1 extras.doc

```
--history--()
AddHotKey()
AllocProgressMeterA()
ArgYesNo()
boopsi_GetGInfo()
Busy()
CopyString()
db_EntryToNN()
db_GetEntryData()
db_NextEntry()
EnqueueName()
ex_CloseLibs()
ex_OpenLibs()
EZReq()
FindLine()
FreeProgressMeter()
ftos()
gui_GhostRect()
gui_MaxStrFontLen()
gui_StrFontLen()
gui_StrLength()
IsWhiteSpace()
key_Shifted()
key_Unshifted()
MultiAllocMemA()
MultiAllocPooledA()
MultiAllocVecA()
MultiFreeMemA()
MultiFreePooledA()
MultiFreeVecA()
NextNNStr()
nns_AddNNStr()
nns_GetNNData()
nns_NNStrLen()
NNString_Overview()
NotBusy()
OBSOLETE--datasheet--()
OBSOLETE_CheckInnerWindowSize()
```

```
  OBSOLETE_CheckWindowSize()
  OBSOLETE_GetGUIScale()
  OBSOLETE_LG_AddGadgets()
  OBSOLETE_LG_CreateGadgets()
  OBSOLETE_LG_FreeGadgets()
  OBSOLETE_LG_GetGadget()
  OBSOLETE_LG_GetGadgetAttrs()
  OBSOLETE_LG_RemoveGadgets()
  OBSOLETE_LG_SetGadgetAttrs()
  OBSOLETE_MakeGadgets()
  OBSOLETEDrawBevelBoxes()
  PD_PackData()
  PD_UnpackData()
  PhraseInStr()
  ProcessTagList()
  RenderText()
  str_Strip()
  StrIStr()
  tag_AddTag()
  tag_AddTags()
  tag_AllocTags()
  tag_ClearNumTags()
  tag_ClearTags()
  tag_CountUserTags()
  tag_FreeTags()
  tag_RemTag()
  tag_TagDone()
  tag_TagMore()
  thread_EndThread()
  thread_PutTMsg()
  thread_PutTMsg_Sync()
  thread_PutTMsg_TagList()
  thread_StartThread()
  UpdateProgressMeterA()
  nns_ProcessNNStr()
```

1.2 extras.lib/--history--

```
Versions
    1.3 - thread_ functions made usable.
```

1.3 extras.lib/AddHotKey

```
NAME
    AddHotKey -- Add a hotkey to a Broker.

SYNOPSIS
    cxobj = AddHotKey(Broker,BrokerPort,HotKey,ID)

    CxObj *AddHotKey(CxObj *,struct MsgPort *,STRPTR,ULONG)

FUNCTION
```

```
    Creates a hotkey for a broker.

INPUTS
    Broker     - Broker CxObj to attach hotkey to.
    BrokerPort - Broker's MsgPort.
    HotKey     - Null terminated string.
    ID         - Hot keys ID.

RESULT
    pointer to a CxObj.

EXAMPLE

NOTES
 requires commodities.library to be opened.
  commodities library already has HotKey()

BUGS

SEE ALSO
```

1.4 extras.lib/AllocProgressMeterA

```
NAME
    AllocProgressMeterA -- Allocate and initialize a progressmeter.
    AllocProgressMeter -- varargs stub for AllocProgressMeterA().

SYNOPSIS
    meter = AllocProgressMeterA(TagList)

    ProgressMeter AllocProgressMeterA( struct TagItem *);

    meter = AllocProgressMeterA(Tag, ... )

    ProgressMeter AllocProgressMeterA( Tag, ...);

FUNCTION
    This function allocates and initializes a ProgressMeter.

INPUTS
    TagList -
      One of the following two are required.
      PM_Screen - The screen to place the meter on.
                  (struct Screen *)
      PM_ParentWindow - The parent window of the meter.
                        (struct Window *)

      PM_MsgPort    - Already existing msgport to send the meter's
                      window events through. (Not implemented)

      PM_TextAttr   - Font to use in the meter. defaults to the
                      screen font. (struct TextAttr *)

      PM_LeftEdge   - Defaults to be centered on PM_ParentWindow
      PM_TopEdge    - or PMScreen.
```

```
        PM_MinWidth   - Set the minimum width of the meter's
                        window.
        PM_MinHeight  - Set the minimum height of the meter's
                        window (Not implemented).

        PM_WinTitle   - Meter's Window title. (STRPTR)

        PM_LowText    - default "0%"   (STRPTR)
        PM_HighText   - default "100%" (STRPTR)

        PM_MeterFormat - A printf style format string used inside the meter.
                         default "%ld%%". (STRPTR)
        PM_MeterType   - How PM_MeterFormat is used,
                           PM_TYPE_PERCENTAGE - uses the percentage of
                             where PM_MeterValue is between PM_LowValue and
                             PM_HighValue for the argument of
                             PM_MeterFormat.
                           PM_TYPE_NUMBER - Uses the meter's value for the
                             argument of PM_MeterFormat.
                           PM_TYPE_STRING - Doens't process the meter's
                             value simply displays the text from
                             PM_MeterFormat
        PM_MeterLabel    - The label above the meter.  default NULL
        PM_MinMeterWidth - The minimum meter bar width, the default minimum
                           is 80

        PM_MeterPen      - default fillpen
        PM_MeterBgPen    - default backgroundpen
        PM_FormatPen     - default highlight text
        PM_MeterLabelPen - default highlight text
        PM_LowTextPen    - default text pen
        PM_HighTextPen   - default text pen

        PM_MeterValue    - (IS) default   0 (LONG)
        PM_LowValue      - (IS) default   0 (LONG)
        PM_HighValue     - (IS) default 100 (LONG)

        PM_Ticks         -  Ticks to draw under the meter box
                              defaults to 0 for none

        PM_CancelButton - Create a Cancel button? (BOOL)
        PM_CancelText   - Text for cancel button. default "Cancel". (STRPTR)
        PM_QueryCancel  - (S) The number of time the user
                          has pressed the cancel button since the last
                          PM_QueryCancel (ULONG *)

        the following three are not implemented
        PM_CancelID - Creates an IDCMP_GADGETUP event when the Cancel button
                      is clicked. IntuiMessage->IAddress will be a pointer
                      to a gadget whose GadgetID is taken from this tag.
                      To be used in conjunctoin with the PM_MsgPort tag.
        PM_CancelSigNum  - Sets a signal when the Cancel button is clicked
        PM_CancelSigTask - Task to signal (struct Task *)

RESULT
    returns a pointer to a ProgressMeter. or NULL on failure.
```

```
   EXAMPLE

   NOTES
     requires diskfont, exec, gadtools, graphics, intuition & utility
     libraries to be open.

   BUGS
       Currently uses SmartRefresh window.

   SEE ALSO
       FreeProgressMeter(), UpdateProgressMeterA()
```

1.5 extras.lib/ArgYesNo

```
   NAME
       ArgYesNo - Get a boolean tooltype value.

   SYNOPSIS
       yes = ArgYesNo(TTypes,Entry,DefVal)

       BOOL ArgYesNo(UBYTE **, STRPTR, DefVal);

   FUNCTION
       This function returns the value of a boolean tooltype.

   INPUTS
       TTypes - a ToolTypes array returned by ArgArrayInit()
       Entry - the entry to search for.
       DefVal - the default boolean value.

   RESULT
       This function only considers the first letter of the
       of the value for the tooltype.  If the first letter
       of the value for the tooltype is 'Y' 'y' 'T' or 't'
       then this function returns 1, if the function finds
       'N' 'n' 'F' or 'f' then it returns 0, if this function
       finds any other character or cannot find the tooltype,
       then the function returns the DefVal.

   NOTES
       must link with amiga.lib

   SEE ALSO
     amiga.lib/ArgArrayInit(), amiga.lib/ArgString(),
     amiga.lib/ArgInt(), amiga.lib/ArgArrayDone()
```

1.6 extras.lib/boopsi_GetGInfo

```
   NAME
       boopsi_GetGInfo -- Get the GadgetInfo pointer from common BOOPSI messa
ges
```

```
SYNOPSIS
    ginfo = boopsi_GetGInfo(Message)

    struct GadgetInfo *boopsi_GetGInfo(Msg);

FUNCTION
    Gets the pointer to the GadgetInfo structure from a BOOPSI
    message.

INPUTS
    Message - BOOPSI message pointer.

RESULT
    pointer to GadgetInfo structure or NULL.

NOTES
    Only handles OM_SET, OM_UPDATE, OM_NOTIFY, GM_HITTEST,
    GM_RENDER, GM_GOACTIVE, GM_HANDLEINPUT, GM_GOINACTIVE
    and GM_LAYOUT methods.

BUGS

SEE ALSO
```

1.7 extras.lib/Busy

```
NAME
    Busy -- displays busy pointer.

SYNOPSIS
    Busy(Window)

    void Busy(struct Window *);

FUNCTION
    Blocks input to a window by opening a requester.

INPUTS
    Window -

RESULT
    None.

NOTES
    requires intuition.library to be open.

    This does not change the Window's IDCMP flags, so
    unless you change them your still going to receive
    input events.
    This function does nothing in WB versions less than
    3.0

SEE ALSO
    NotBusy()
```

1.8 extras.lib/CopyString

```
NAME
    CopyString -- Copy a string

SYNOPSIS
    newstring = CopyString(Source, MemFlags)

    STRPTR CopyString(STRPTR, ULONG);

FUNCTION
    Allocates memory using AllocVec and copies a string.

INPUTS
    Source - the source string to copy
    MemFlags - Memory flags see exec.library/AllocVec()

RESULT
    String pointer or NULL.

NOTES
    newstring must be freed with FreeVec.

SEE ALSO
  exec.library/AllocVec(), exec.library/FreeVec()
```

1.9 extras.lib/db_EntryToNN

```
NAME
    db_EntryToNN -- Retrieve data from an ENTRY of a database.

SYNOPSIS
    STRPTR db_EntryToNN(BPTR File, STRPTR EntryName)

FUNCTION
    Get data from a Database, see db_GetEntryData().
    All the data in the entry is put into an NNString.
    The data can then be parsed with nns_GetNNData()
```

1.10 extras.lib/db_GetEntryData

```
NAME
    db_GetEntryData -- Retrieve data from an ENTRY of a database.

SYNOPSIS
    db_GetEntryDataA(File, EntryName, Items) -- NOT implemented --

    BOOL db_GetEntryDataA(BPTR, STRPTR struct EItem *);

    db_GetEntryData(File, EntryName, Items, ... )

    BOOL db_GetEntryData(BPTR, STRPTR, STRPTR, ... );
```

```
FUNCTION
    Retrieve data from a simple ENTRY based database.

INPUTS
    File - an AmigaDOS BPTR to a file.
    Name - an array of struct Etems, the last struct EItem should have
           it's Name field set to NULL. (see example)
RESULT
    returns 0 on failure, possibly due to EOF, improper file format, or
    lack of memory.

    To support multiple occurances of an item Name in an Entry, this
    function returns NNStrings.  The strings are stored end to end in
    the order that they were read from the file.

    On success, each EItem.ReturnString either points to an NNString,
    or NULL if no data for that Name was found.

    On failure, all EItem.ReturnStrings are NULL, and any data collected
    is freed.

EXAMPLE
    STRPTR title,desc;
    BPTR File;

    if(db_GetEntryData(File,"ENTRY",
                        "TITLE"  ,&title,
                        "DESC"   ,&desc,
                           0))
    {
      if(title)
      {
        printf("%s - ",title);
        FreeVec(title);
      }

      if(desc)
      {
        printf("%s\n");
        FreeVec(desc);
      }
    }


NOTES
    The database file format is an ASCII text file, and consists
    of "ENTRY"'s, that look like this:

    <ENTRYNAME>
    {
      <ITEMNAME> = <data>
      <ITEMNAME> = <data>
    }

    an example file format from above might be:
```

```
    ENTRY
    {
      TITLE=Cows 'R Us
      DESC=All you want to know about beef.
    }

    case of <ENTRYNAME> and <ITEMNAMES> is not important.
    the equal sign is required.

HISTORY
    This code probably isn't all that usefull, but it was the
    code behind the database of now defunct Tampa Bay Amiga Group's
    Amiga Support Directory.  Unfortunately, TBAG died before the
    ASD could begine to grow, and I haven't used this code since.

BUGS
    Not reentrant.

SEE ALSO
    nns_ProcessNNStr, nns_AddNNStr(), nns_NextNNStr(), db_EntryToNN()
```

1.11 extras.lib/db_NextEntry

```
NAME
    db_NextEntry - Find the next entry in a database file.

SYNOPSIS
    error = db_NextEntry(File, EntryName, Buffer, BufferSize)

    LONG db_NextEntry(BPTR, STRPTR, STRPTR, ULONG);

FUNCTION
    Seeks for the next entry in a database file.

INPUTS
    File - AmigaDos file handle.
    EntryName - the name of the entry, usually "ENTRY".
           case is insignificant.
    Buffer - a buffer for reading data from a file.
    BufferSize - the size of the buffer.

RESULT
    returns 0 on failure.  on success the file is positioned
    inside the entry.

EXAMPLE

NOTES
    This function is mainly used for other lib functions

BUGS

SEE ALSO
```

1.12 extras.lib/EnqueueName

```
NAME
    EnqueueName -- Place a Node in a sorted List.

SYNOPSIS
    EnqueueName(List,Node)

    void EnqueueName(struct List *,struct Node*)

FUNCTION
    Place a Node in a sorted List prioritized by Node.ln_Name
    and ln_Pri.

INPUTS
    List - pointer to a List to place Node into.
    Node - pointer to a Node to be placed in the List.

RESULT
    None.

NOTES
    The List must be presorted by ln_Name and ln_Pri.
    Every node must have its ln_Name field pointing to
    a NULL terminated string.
```

1.13 extras.lib/ex_CloseLibs

```
NAME
    ex_CloseLibs -- close multiple libraries.

SYNOPSIS
    ex_CloseLibs(Libs)

    void ex_CloseLibs(struct Libs *);

FUNCTION
    Close multiple libraries using the same array of
    struct Libs as used in OpenLibs.

INPUTS
    Libs - A pointer to an array of struct Libs.

RESULT
    none.

NOTES
    exec.library must already be opened.(usually done by the
    compiler's startup code)

    revision 1.1
      corrected autodoc.
      now openes ExecBase on it's own.
    revision 1.2
```

```
      renamed to ex_CloseLib due to conflict with reaction.lib

BUGS

SEE ALSO
    ex_OpenLibs()
```

1.14 extras.lib/ex_OpenLibs

```
NAME
    ex_OpenLibs -- attempt to open multiple libraries.

SYNOPSIS
    success = ex_OpenLibs(Argc,ProgName,ErrorStr,
                LibVerFmt,ButtonText,Libs)

    BOOL ex_OpenLibs(ULONG Argc, STRPTR ProgName, STRPTR ErrorString,
           STRPTR LibVerFmt, STRPTR ButtonText, struct Libs *Libs);

FUNCTION
    Attempt to open multiple Libraries.  If any library
    fails to open, a requester is opened to notify the
    user listing all of the libraries that failed to open.

INPUTS
    Argc       - argc from main()
    ProgName   - pointer to a string containing the
                 program's name.
    ErrorStr   - error string.  May be NULL.  Defaults to
                 "The following libraries are required:\n";
    LibVerFmt  - printf style format string.  Defaults to
                  "  %s version %ld\n"
    ButtonText - If a requester is used to display an error
                  message, this text is used in the button.
                  Defaults to "Ok"
    Libs       - an array of libraries to open.

RESULT
    return 0 on failure and non-zero on success.

EXAMPLE
    struct IntuitionBase *IntuitionBase;
    struct GfxBase *GfxBase;
    struct Library *GadToolsBase;

    struct Libs MyLibs[]=
    {
      &IntuitionBase,"intuition.library"    ,37, 0,
      &GfxBase      ,"graphics.library"     ,37, 0,
      &GadToolsBase ,"gadtools.library"     ,37, 0,
      &LocaleBase   ,"datatypes.library"    ,39, OLF_OPTIONAL,
      0,0,0
    };

    void main(int argc, char **argv)
```

```
    {
      if(ex_OpenLibs(arcg,"MyProgram",0,0,0,MyLibs))
      {
        ...
        CloseLibs(MyLibs);
      }
    }

NOTES
    On error, this function will automatically display an
    intuition requester if Argc=0 or print error information
    out to STDIO if Argc>0.

    exec.library must already be open.(usually done by the
    compiler's startup code)

    revision 1.1
      autodoc fix
      now opens exec.library on it's own.
    revision 1.2
      renamed to ex_OpenLib due to conflict with reaction.lib



BUGS

SEE ALSO
    ex_CloseLibs()
```

1.15 extras.lib/EZReq

```
NAME
    EZReq -- create an Intuition EasyRequest.

SYNOPSIS
    retval = EZReq(Win, IDCMP_ptr, Title, Text,
                   ButtonText,Arg,...)

    LONG EZReq(struct Window *, ULONG *, STRPTR, STRPTR,
               STRPTR, ULONG, ...);

FUNCTION
    This function provides an easier method to use the
    intuition/EasyRequestArgs() function.

INPUTS
    Win -
    IDCMP_ptr -
    Title -
    Text -
    ButtonText -
    Arg -

RESULT
```

```
EXAMPLE

NOTES
    requires the exec.library to be open, automatically opens
    intuition.library.

BUGS

SEE ALSO
```

1.16 extras.lib/FindLine

```
NAME
    FindLine - find the next matching line in a file.

SYNOPSIS









FUNCTION


INPUTS

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.17 extras.lib/FreeProgressMeter

```
NAME
    FreeProgressMeter -- Close a PregressMeter.

SYNOPSIS
    FreeProgressMeter(PM)

    void FreeProgressMeter(ProgressMeter );

FUNCTION
    Close and deallocated a ProgressMeter.

INPUTS
    PM - pointer to an existing ProgressMeter or NULL.
```

```
RESULT
    none.

EXAMPLE

NOTES
  requires diskfont, exec, gadtools, graphics, intuition & utility
  libraries to be open.

BUGS

SEE ALSO
    AllocProgressMeterA(), UpdateProgressMeterA()
```

1.18 extras.lib/ftos

```
NAME
    ftos - Convert a float to a string

SYNOPSIS




FUNCTION


INPUTS

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.19 extras.lib/gui_GhostRect

```
NAME
    GhostRect -- Cover a rectangular are with
                 a ghosted pattern.

SYNOPSIS
    gui_GhostRect(RP, Pen, X0, Y0, X1, Y1)

    void gui_GhostRect(struct RastPort *, ULONG,
```

```
          WORD, WORD, WORD, WORD);

FUNCTION
    Covers a rectangular area with a ghosted pattern
    using the specified pen number.

INPUTS
    RP - Pointer to a RastPort.
    Pen - the pen number to use for the pattern.
    X0 - left edge
    Y0 - top edge
    X1 - right edge
    Y1 - bottom edge
```

1.20 extras.lib/gui_MaxStrFontLen

```
NAME
    gui_MaxStrFontLen - get the maximum pixel length of a
    string containg N characters.

SYNOPSIS
    length=gui_MaxStrFontLen(Font, Chars, LowChar, HighChar)

    LONG gui_MaxStrFontLen(struct TextFont *, ULONG, UBYTE, UBYTE);

FUNCTION
    This function returns the maximum number of pixels
    a string with a certain number of characters could
    occupy.

INPUTS
    Font - struct TextFont * previously opened by
           OpenFont() or OpenDiskFont().
    Chars - the max number characters in a string.
    LowChar - the ASCII number of lowest character
              that could be in a string.
    HighChar - the ASCII number of the highest character
               that could be in a string.

EXAMPLE
     find the longest pixel length of a number
     upto 3 digits
    long maxnumberlen;
    struct TextFont *tf;

    maxnumberlen=gui_MaxStrFontLen(tf,3,'0','9');

RESULT
    the pixel length or 0 if the Font parameter
    is NULL.
```

1.21 extras.lib/gui_StrFontLen

```
NAME
    gui_StrFontLen - get the pixel length of a string.

SYNOPSIS
    length=gui_StrFontLen(Font, Str)

    LONG gui_StrFontLen(struct TextFont *,STRPTR);

FUNCTION
    This function returns the number of pixels a string
    would occupy if rendered in the specified Font.

INPUTS
    Font - struct TextFont * previously opened by
           OpenFont() or OpenDiskFont().
    Str - a pointer to a null terminated string.

RESULT
    the pixel length of the string or 0 if either parameter
    is NULL.
```

1.22 extras.lib/gui_StrLength

```
NAME
    gui_StrLength -- Get the pixel length of a string.

SYNOPSIS
    Len = gui_StrLength(Tags)

    LONG gui_StrLength(Tag, ... );

FUNCTION
    Find the length of a given string.  If multiple strings
    are given, returns the longest length.

INPUTS
    Tags
      SL_TextFont - (struct TextFont *)The font the string will
        be rendered in. (required)
      SL_IgnoreChars - (STRPTR) Null terminated string of characters
        to ignore from the length calculation, useful for underscores
        in gadget text, etc.
      SL_String - (STRPTR) String to size.

RESULT
    Length in pixels of longest string.

EXAMPLE
      *find length of button text without the "_"
      len= gui_StrLength(SL_TextFont     ,TF,
                     SL_String       ,"_Button",
                     SL_IgnoreChars  ,"_",
                     TAG_DONE);
```

```
      * find maximum length of red, green and blue without "_"
      len= gui_StrLength(SL_TextFont     ,TF,
                     SL_String       ,"_Red",
                     SL_String       ,"_Green",
                     SL_String       ,"_Blue",
                     SL_IgnoreChars  ,"_",
                     TAG_DONE);

NOTES

BUGS

SEE ALSO
```

1.23 extras.lib/IsWhiteSpace

```
NAME
    IsWhiteSpace -- Is a character a "white-space"

SYNOPSIS
    IsWhiteSpace(Char)

    BOOL IsWhiteSpace(char);

FUNCTION
    Indicate whether or not a character is a "white-space"

INPUTS
    Char - a letter.

RESULT
    non-0 if Char is " "(space), "\t"(tab), or "\n"(cr)

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.24 extras.lib/key_Shifted

```
NAME
    key_Shifted -- Get shifted character of suppied character

SYNOPSIS
    shiftedchar = key_Shifted(character)

    ULONG key_Shifted(char);

FUNCTION
```

```
    Returns the shifted character for the supplied character.
    For example (on the USA keymap)
      key_Shifted('#') = '#
      key_Shifted('3') = '#

NOTE
    This function was previously KeyShifted()

SEE ALSO
    Key_Unshifted()
```

1.25 extras.lib/key_Unshifted

```
NAME
    key_Unshifted -- Get unshifted character of suppied character

SYNOPSIS
    unshiftedchar = key_Unshifted(character)

    ULONG key_Unshifted(char);

FUNCTION
    Returns the unshifted character for the supplied character.
    For example (on the USA keymap)
      key_Unshifted('#') = '3'
      key_Unshifted('3') = '3'

NOTE
    This function was previously KeyUnshifted()

SEE ALSO
    KeyShifted()
```

1.26 extras.lib/MultiAllocMemA

```
NAME
    MultiAllocMemA -- Multiple AllocMem.
    MultiAllocMem -- varargs stub.

SYNOPSIS
    succes = MultiAllocMemA(Flags, MemTagList)

    BOOL MultiAllocMem(ULONG, struct MemTag *);

    succes = MultiAllocMemA(Flags, MemTag)

    BOOL MultiAllocMemA(Flags, ULONG, ...);

FUNCTION
    Attempt to allocate one or more memory chunks
    using AllocMem.
```

```
INPUTS
    Flags - MA_FAILSIZE0: fail all allocations if any
            have a size of 0.  if your application will be
            allocating memory of dynamic sizes, and if
            you want allocations of 0 bytes to fail, then
            set this flag.
    MemTag - pointer to an array of struct MemTag.
               vt_Ptr is the address of a pointer.
               vt_Size is the size of the allocation.
               vt_MemFlags are the exec memory (MEMF_) flags.
             Last tag should have vt_Ptr = NULL.

RESULT
    zero if it couldn't allocate the requested memory. or non-zero
    on success. vt_Ptrs will be point to a allocated
    memory chunk or NULL.

EXAMPLE
    EX1:
      struct foo *bar;
      STRPTR dest;
      APTR cow;

     if( MultiAllocMem(0,
                       &bar,  sizeof(struct foo),  MEMF_CLEAR,
                       &dest, 25,                  MEMF_PUBLIC,
                       &cow,  100,                 MEMF_FAST|MEMF_PUBLIC,
                       0))
      {
        ...
        MultiFreeMem(3,
                       bar  ,sizeof(struct foo),
                       dest ,25,
                       cow  ,100);
      }

    EX2: This will never fail.
      APTR foo;
      if(MultiAllocMem(0,
                       &foo,0,MEMF_CLEAR,
                       0)
      {...}

    EX3: This will always fail.
      APTR foo;
      if(MultiAllocMem(MA_FAILSIZE0,
                       &foo,0,MEMF_CLEAR,
                       0)
      {...}

NOTES
    requires exec.library to be open.

    if the MA_FAILSIZE0 Flag is not set, 0 byte allocations
    will pass even though no memory will be allocated for that.
    entry and mt_Ptr will be set to 0.
```

```
    The memory allocated may be freed individually with
    exec.library/FreeMem()

BUGS

SEE ALSO
    MultiAllocVecA(), MultiFreeVecA(), MultiFreeMemA(),
    MultiAllocPooledA(), MultiFreePooledA(),
    exec.library/AllocVec(), exec.library/FreeVec()
    exec.library/AllocMem(), exec.library/FreeMem()
    exec.library/AllocPooled(), exec.library/FreePooled()
```

1.27 extras.lib/MultiAllocPooledA

```
NAME
    MultiAllocPooledA -- Multiple AllocPooled.
    MultiAllocPooled -- varargs stub.

SYNOPSIS
    succes = MultiAllocPooledA(Flags, PoolTagList)

    BOOL MultiAllocPooled(ULONG, struct PoolTag*);

    succes = MultiAllocPooledA(Flags, PoolTag)

    BOOL MultiAllocPooledA(Flags, ULONG, ...);

FUNCTION
    Attempt to allocate one or more memory chunks
    using AllocPooled.

INPUTS
    Flags - MA_FAILSIZE0: fail all allocations if any
            have a size of 0.  if your application will be
            allocating memory of dynamic sizes, and if
            you want allocations of 0 bytes to fail, then
            set this flag.
    PoolTag - pointer to an array of struct PoolTag.
              pt_Ptr is the address of a pointer.
              pt_Size is the size of the allocation.
             Last tag should have vt_Ptr = NULL.

RESULT
    zero if it couldn't allocate the requested memory. or non-zero
    on success. In either case, vt_Ptrs will be point to a allocated
    memory chunk or NULL.

EXAMPLE
    EX1:
      APTR pool;
      struct foo *bar;
      STRPTR dest;
      APTR cow;

      if(pool=CreatePool(MEMF_ANY,300,300))
```

```
      {
        if( MultiAllocPooled(pool,0,
                        &bar,  sizeof(struct foo),
                        &dest, 50,
                        &cow,  100,
                        0))
        {
          ...
          MultiFreePooled(pool,3,
                           bar,  sizeof(struct foo),
                           dest, 50,
                           cow,  100);
        }
        DeletePool(pool);
      }

    EX2: This will never fail.
      if(MultiAllocPooled(pool,0,
                        &foo, 0,
                        0)
      {...}

    EX3: This will always fail.
      if(MultiAllocPooled(pool,MA_FAILSIZE0,
                        &foo, 0,
                        0)
      {...}

NOTES
    requires exec.library to be open.

    if the MA_FAILSIZE0 flag is not set, 0 byte allocations
    will pass even though no memory will be allocated for that.
    entry and mt_Ptr will be set to 0.

    The memory allocated may be freed individually with
    exec.library/FreePooled()

BUGS

SEE ALSO
    MultiFreeVecA(), MultiAllocMemA(),MultiFreeMemA(),
    MultiFreePooledA(),
    exec.library/AllocVec(), exec.library/FreeVec()
    exec.library/AllocMem(), exec.library/FreeMem()
    exec.library/AllocPooled(), exec.library/FreePooled()
```

1.28 extras.lib/MultiAllocVecA

```
NAME
    MultiAllocVecA -- Multiple AllocVec.
    MultiAllocVec -- varargs stub.

SYNOPSIS
    succes = MultiAllocVecA(Flags, VecTagList)
```

```
    BOOL MultiAllocVec(ULONG, struct VecTag *);

    succes = MultiAllocVecA(Flags, VecTag)

    BOOL MultiAllocVecA(Flags, ULONG, ...);

FUNCTION
    Attempt to allocate one or more memory chunks
    using AllocVec.

INPUTS
    Flags - MA_FAILSIZE0: fail all allocations if any
            have a size of 0.  if your application will be
            allocating memory of dynamic sizes, and if
            you want allocations of 0 bytes to fail, then
            set this flag.
    VecTag - pointer to an array of struct VecTag.
              vt_Ptr is the address of a pointer.
              vt_Size is the size of the allocation.
              vt_MemFlags are the exec memory (MEMF_) flags.
             Last tag should have vt_Ptr = NULL.

RESULT
    zero if it couldn't allocate the requested memory. or non-zero
    on success. In either case, vt_Ptrs will be point to a allocated
    memory chunk or NULL.

EXAMPLE
    EX1:
      struct foo *bar;
      STRPTR dest;
      APTR cow;

     if( MultiAllocVec(0,
                       &bar,  sizeof(struct foo),  MEMF_CLEAR,
                       &dest, strlen(str)+1,       MEMF_PUBLIC,
                       &cow,  100,                 MEMF_FAST|MEMF_PUBLIC,
                       0))
      {
        ...
        MultiFreeVec(3,bar,dest,cow);
      }

    EX2: This will never fail.
      APTR foo;
      if(MultiAllocVec(0,
                       &foo,0,MEMF_CLEAR,
                       0)
      {...}

    EX3: This will always fail.
      APTR foo;
      if(MultiAllocVec(MA_FAILSIZE0,
                       &foo,0,MEMF_CLEAR,
                       0)
      {...}
```

```
NOTES
    requires exec.library to be open.

    if the MA_FAILSIZE0 flag is not set, 0 byte allocations
    will pass even though no memory will be allocated for that.
    entry and mt_Ptr will be set to 0.

    The memory allocated may be freed individually with
    exec.library/FreeVec()

BUGS

SEE ALSO
    MultiFreeVecA(), MultiAllocMemA(),MultiFreeMemA(),
    MultiAllocPooledA(),MultiFreePooledA(),
    exec.library/AllocVec(), exec.library/FreeVec()
    exec.library/AllocMem(), exec.library/FreeMem()
    exec.library/AllocPooled(), exec.library/FreePooled()
```

1.29 extras.lib/MultiFreeMemA

```
NAME
    MultiFreeMemA -- Free multiple memory chunks.
    MultiFreeMem -- varargs stub.

SYNOPSIS
    MultiFreeMemA(Args, FreeTagList)

    void MultiFreeMemA(ULONG, struct FreeTag *);

    MultiFreeMem(Args, FreeTag, ... )

    void MultiFreeMem(ULONG, ULONG, ... );

FUNCTION
    Free multiple memory blocks allocated with MultiAllocMemA()
    or exec.library/AllocMem().

INPUTS
    Args - Number of blocks that are to be freed.
    FreeTagList - An array of FreeTags. may be NULL.
                  ft_Ptr - contains a pointer to a
                           memory block or NULL.
                  ft_Size - the size of the block.

RESULT
    none.

NOTES
    requires exec.library to be open.

SEE ALSO
    MultiAllocVecA(), MultiFreeVecA(), MultiAllocMemA(),
    MultiAllocPooledA(), MultiFreePooledA(),
```

```
    exec.library/AllocVec(), exec.library/FreeVec()
    exec.library/AllocMem(), exec.library/FreeMem()
    exec.library/AllocPooled(), exec.library/FreePooled()
```

1.30 extras.lib/MultiFreePooledA

```
NAME
    MultiFreePooledA -- Free multiple memory chunks.
    MultiFreePooled -- varargs stub.

SYNOPSIS
    MultiFreePooledA(Pool, Args, FreeTagList)

    void MultiFreePooledA(APTR, ULONG, struct FreeTag *);

    MultiFreePooled(Pool, Args, FreeTag, ... )

    void MultiFreePooled(APTR, ULONG, ULONG, ... );

FUNCTION
    Free multiple memory blocks allocated with MultiAllocPooledA()
    or exec.library/AllocPooled().

INPUTS
    Args - Number of blocks that are to be freed.
    FreeTagList - An array of FreeTags. may be NULL.
                  ft_Ptr - contains a pointer to a
                           memory block or NULL.
                  ft_Size - the size of the block.

RESULT
    none.

NOTES
    requires exec.library to be open.

SEE ALSO
    MultiAllocVecA(), MultiFreeVecA(), MultiAllocMemA(),
    MultiAllocPooledA(), MultiFreePooledA(),
    exec.library/AllocVec(), exec.library/FreeVec()
    exec.library/AllocMem(), exec.library/FreeMem()
    exec.library/AllocPooled(), exec.library/FreePooled()
```

1.31 extras.lib/MultiFreeVecA

```
NAME
    MultiFreeVecA -- Free multiple memory chunks.
    MultiFreeVec -- varargs stub.

SYNOPSIS
    MultiFreeVecA(Args,MemBlockList)
```

```
    MultiFreeVecA(ULONG, APTR *);

    MultiFreeVec(Args, MemBlock)

    void MultiFreeVec(ULONG, ULONG, ... );

FUNCTION
    Free multiple memory blocks allocated with MultiAllocVecA()
    or exec.library/AllocVec().

INPUTS
    Args - Number of blocks that are to be freed.
    MemBlockList - An array of pointers to memory
                   blocks to be freed or NULL.

RESULT
    none.

NOTES
    requires exec.library to be open.

SEE ALSO
    MultiAllocVecA(), MultiFreeVecA(), MultiAllocMemA(), MultiFreeMemA(),
    MultiAllocPooledA(), MultiFreePooledA(),
    exec.library/AllocVec(), exec.library/FreeVec()
    exec.library/AllocMem(), exec.library/FreeMem()
    exec.library/AllocPooled(), exec.library/FreePooled()
```

1.32 extras.lib/NextNNStr

```
NAME
    NextNNStr -- Get the next string in a double NULL terminted
                 string array (aka NNString).

SYNOPSIS
    nextstring=NextNNStr(NNString)

    STRPTR NextNNStr(STRPTR);

FUNCTION
    returns a pointer to the next string contained in a
    double NULL teminated string array, or NULL.

INPUTS
    NNString - A pointer to some part of a double NULL string.

RESULT
    Pointer to the next string or NULL if there is no string.

EXAMPLE
    ** this example steps through an NNString. **

    STRPTR NNString;
    STRPTR str;
```

```
    for(str=NNString; str; str=NextNNStr(str))
    {
      printf("%s\n",str);
    }

    The above can be simplified by using the ProcessNNStr macro
    defined in extras/nnstring.h

    STRPTR NNString;
    STRPTR str;

    ProcessNNStr(NNString,str)
    {
      printf("%s\n",str);
    }

NOTES

BUGS

SEE ALSO
    NNString AddNNStr() ProcessNNStr, extras/macros.h
```

1.33 extras.lib/nns_AddNNStr

```
NAME
    nns_AddNNStr -- append a normal string to a NNString.

SYNOPSIS
    NewNNStr=nns_AddNNStr(NNStr,New)

    STRPTR nns_AddNNStr(STRPTR, STRPTR);

FUNCTION


INPUTS
    NNStr - an existing NNString or NULL, if NULL
        the function converts str into an NNString.
    New - a regular NULL terminated string.

RESULT
    A new NNString or NULL, NNStr *WILL* be freed
    regardless of result.  if New is NULL, NNStr will
    be returned.

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.34 extras.lib/nns_GetNNData

```
NAME
    nns_GetNNData -- Find data in a NNString

SYNOPSIS
    STRPTR GetNNData(STRPTR NNStr, STRPTR Name, STRPTR DefVal);

FUNCTION
    Retrieves data from a NNString.

    nnstr="TITLE=Joe\0COLOR=Green\0TITLE=Test\0\0"

    t=GetNNData(nnstr,"TITLE","None");

    t will equal "Joe\0Test\0\0"

SEE ALSO
```

1.35 extras.lib/nns_NNStrLen

```
NAME
    nns_NNStrLen -- Return the length of an NNString.

SYNOPSIS
    length=nns_NNStrLen(NNStr)

    LONG nns_NNStrLen(STRPTR);

FUNCTION
    Returns the length in bytes of an NNString, including the
    trailing NULLs.

INPUTS
    NNStr - NNString pointer.

RESULT
    Size of NNStr include NULLs.

SEE ALSO
```

1.36 extras.lib/NNString_Overview

```
FUNCTION
    An NNString is an array of strings, stored end to end, ending
    with a double NULL.  Individual strings are seperated by NULLs.

EXAMPLE
    An NNString representing this array of strings:
    "Cow" "Dog" "Barn"
    would look like this:
    "Cow\0Dog\0Barn\0\0"
```

```
   SEE ALSO
       NextNNStr(), ProcessNNStr, GetEntryData(), AddNNStr(),
       NNStrLen()
```

1.37 extras.lib/NotBusy

```
   NAME
       NotBusy -- displays busy pointer.

   SYNOPSIS
       NotBusy(Window)

       void NotBusy(struct Window *);

   FUNCTION
       Removes the busy pointer for the specified
       window.

   INPUTS
       Window - the window to remove the busy pointer
                from.

   RESULT
       None.

   NOTES
       requires intuition.library to be open.

       This does not change the Window's IDCMP flags.
       This function does nothing in WB versions less than
       3.0

   SEE ALSO
       Busy()
```

1.38 extras.lib/OBSOLETE--datasheet--

```
   NOTES
       I deem some functions obsolete.

       All the GadTools layout code and support Functions are obsolete,
       due to OS3.5 being released with Reaction/ClassAct.

       I found it easier to convert a gui than to maintain it with my code :)
       They remain in the archive and in the .lib's only because some program
s
       still need them to compile.
```

1.39 extras.lib/OBSOLETE_CheckInnerWindowSizeras.lib/OBSOLETE_CheckInnerWind

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    CheckInnerWindowSize - See if a window will fit on a screen.

SYNOPSIS
    windowfits = CheckInnerWindowSize(Scr, Width, Height,
                                      XScale, YScale)

    BOOL CheckInnerWindoeSize(struct Screen *, WORD, WORD,
                                    float, float);

FUNCTION
    This function checks to see if a window with the specified
    inner dimensions will fit on a screen.

INPUTS
    Scr - the Screen the window is destine for.
    Width -  the base inner width of the window.
    Height - the base inner height of the window.
    XScale - the proposed x scale of the window.
    YScale - the proposed y scale of the window.

RESULT
    Returns TRUE if the window will fit, and FALSE if not.
    if the XScale or YScale is <=0 it will also fail.

SEE ALSO
    GetGUIScale(), CheckWindowSize()
```

1.40 extras.lib/OBSOLETE_CheckWindowSize

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    CheckWindowSize - See if a window will fit on a screen.

SYNOPSIS
    windowfits = CheckWindowSize(Scr, Width, Height,
                                    XScale, YScale)

    BOOL CheckWindowSize(struct Screen *, WORD, WORD,
                                    float, float);

FUNCTION
    This function checks to see if a window will fit on a screen.

INPUTS
    Scr - the Screen the window is destine for.
    Width - the base width of the window.
    Height - the base height of the window.
```

```
    XScale - the proposed x scale of the window.
    YScale - the proposed y scale of the window.

RESULT
    Returns TRUE if the window will fit, and FALSE if not.
    if the XScale or YScale is <=0 it will also fail.

SEE ALSO
    GetGUIScale(), CheckInnerWindowSize()
```

1.41 extras.lib/OBSOLETE_GetGUIScale

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    GetGUIScale -- Find the appropriate scale for an
                   interface.

SYNOPSIS
    success = GetGUIScale(TAttr, Strings, &XScale, &YScale)

    BOOL GetGUIScale(struct TextAttr *, struct GUI_String *,
                     float *, float *);

FUNCTION
    This function figures out the minimum size an interface
    should be so that all of the Strings will fit.

INPUTS
    TAttr - The TextAttr to be used for the strings.
    Strings - A NULL terminated array of GUI_String containing
              the Strings to used in the interface and the maximum
              size each string can be before the interface needs
              to be enlarged.
    XScale - the address of a float to contain the X scale factor.
    YScale - the address of a float to contain the Y scale factor.

RESULT
    This function returns non-zero on success, and will have the X &
    YScale values set apropriately.  Returns NULL on failure if the
    font specified in TAttr can not be opened, and X & YScale will be
    set to -1.

NOTES
    requires diskfont.library to be open.

    The reason this function sets X & YScale to -1 on failure is to
    also cause CheckWindowSize() and CheckInnerWindowSize to fail.
    This way you can simply:
    GetGUIScale(ta,strings,&xscale,&yscale)
    if(!CheckWindwoWidth(scr,winwidth,winheight,xscale,yscale))
    { ... revert to topaz.8 ...
    }
```

```
    The GUI_String specifies the maximum size a string can be before
    the interface should be scaled horizontally.  For example, if you
    have a BUTTON_KIND gadget that is 100 pixels wide, then you may
    want to set the maximum size for the string in that gadget to 90.
    (ie.
         struct GUI_String gs[]=
         {
           "Button Text", 90,
           0,0
         };
    )

SEE ALSO
    MakeGadgets()
```

1.42 extras.lib/OBSOLETE_LG_AddGadgets

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    LG_AddGadgets - Add gadgets to a window.

SYNOPSIS









FUNCTION


INPUTS

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.43 extras.lib/OBSOLETE_LG_CreateGadgets

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    LG_CreateGadgets -- Create multiple Gadtools gadgets.
```

```
SYNOPSIS
    gcontrol = LG_CreateGadgets(Tags)

    struct LG_Control *LG_CreateGadgets(Tag Tags, ...);

FUNCTION
    Creates multiple Gadtools, with some layout options.

INPUTS
    Tags (see <extras/layougtgt.h>)

      LG_LeftEdge, LG_XPos - Position of gadget
      LG_TopEdge,  LG_YPos - Position of gadget
          The position of the gadget will depend on these attributes
          and the LG_Justification attribute.  Also see the relational
          macros in <extras/layoutgt.h>
      LG_Width  - Gadget width
      LG_Height - Gadget height

        for the following see the NewGadget structure in
        <libraries/gadtools.h>
      LG_GadgetText
      LG_TextAttr
      LG_GadgetID
      LG_Flags
      LG_VisualInfo
      LG_UserData
      LG_GadgetKind
      LG_GadgetTags
      LG_GadgetTagList

      LG_OffsetX - global offset from left of window
      LG_OffsetY - global offset from top of window
         These two set to topleft of the domain.
         Also these will be added to LG_UseScreenOffsets
         or LG_UseWindowOffsets if set.

      LG_LabelFlags - see LGLF_

      LG_ScaleX - Multiply gadget width and position by these.
                  (percent * 65536)
      LG_ScaleY - Multiply gadget height and position by these.
                  (percent * 65536)

      LG_Justification - Sets the "handle" of the gadget.

      LG_UseScreenOffsets - Sets global offsets based on Window border
                        dimensions specified in the Screen structure
      LG_UseWindowOffsets - Sets global offsets based on Window border
                        dimensions

      LG_EraseRemoved - If set gadgets erase themselves when they are
                        removed using LG_RemoveGadgets

      LG_KeyClass  - Not used
      LG_KeyString - A string of characters that "activate" that
```

```
            gadget, if not specified, LG_CreateGadget() will scan
            the gadget label for the appropriate string.  This
            attribute is cleared after each LG_CreateGadget
             attribute
        LG_ErrorCode - Not used

        LG_Bounds - set offsets or domain area for gadgets
        LG_BoundsLeft   - Left of domain. (Alias for LG_OffsetX)
        LG_BoundsTop    - Top of domain.  (Alias for LG_OffsetY)
        LG_BoundsWidth  - Width of domain.
        LG_BoundsHeight - Height of domain.

        LG_RelHorizGap - Gap between certain relative operations
        LG_RelVertGap  - Gap between certain relative operations

        LG_HorizCells - Sets the number of columns in a table.
        LG_VertCells - Sets the number of rows in a table.
          Sets the number of cells in a table.  The table's pixel size
          is the current LG_Bound size.  These attributes work in
          conjunction with the LG_REL_CELL_ macros defined in
          <extras/layoutgt.h>

        LG_SkipGadgets - Skip the next ti_Data LG_CreateGadgets and
          all Tags in between.

RESULT
    returns NULL on failure, or a pointer the LG_Control structure
    defined in <extras/layoutgt.h>

EXAMPLE
    See the supplied example, if there are any.

NOTES
    Gadgets created by LG_CreateGadgets() Must be freed using
    LG_FreeGadgets OR Removed from it's Window using
    LG_RemoveGadgets *BEFORE* the window is closed

BUGS

SEE ALSO
    LG_FreeGadgets, LG_RemoveGadgets, LG_AddGadgets, LG_RemoveGadgets,
```

1.44 extras.lib/OBSOLETE_LG_FreeGadgets

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    LG_FreeGadgets - free gadgets allocated be LG_CreateGadgets.

SYNOPSIS
    LG_FeeeGadgets(Control)

    void LG_FreeGadgets(struct LG_Control);
```

```
FUNCTION
    deallocates gadgets ans support structures allocated by
     LG_CreateGadget

INPUTS
    Con - pointer to structure returned by LG_CreateGadgets.

RESULT

EXAMPLE

NOTES
    Gadgets created by LG_CreateGadgets() Must be freed using
    LG_FreeGadgets OR Removed from it's Window using
    LG_RemoveGadgets *BEFORE* the window is closed

BUGS

SEE ALSO
```

1.45 extras.lib/OBSOLETE_LG_GetGadget

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    LG_GetGadget - get a gadget pointer using Gadget ID.

SYNOPSIS









FUNCTION


INPUTS

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.46 extras.lib/OBSOLETE_LG_GetGadgetAttrs

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    LG_GetGadgetAttrs

SYNOPSIS




FUNCTION


INPUTS

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.47 extras.lib/OBSOLETE_LG_RemoveGadgets

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    LG_RemoveGadgets - remove gadgets from a window.

SYNOPSIS




FUNCTION


INPUTS

RESULT

EXAMPLE

NOTES
```

```
BUGS

SEE ALSO
```

1.48 extras.lib/OBSOLETE_LG_SetGadgetAttrs

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    LG_SetGadgetAttrs - set gadget attrs.

SYNOPSIS




FUNCTION


INPUTS

RESULT

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.49 extras.lib/OBSOLETE_MakeGadgets

```
OBSOLETE
    Since OS3.5 uses Reaction/ClassAct - this code is obsolete

NAME
    MakeGadgets -- Minimal layout Gadtools gadgets.

SYNOPSIS
    Gadget = MakeGedgets(Scr, VisualInfo, NumGadgets, NewGads,
             NewGadTags, NewGadKinds, Gadgets, TextAttr, XScale,
             YScale)

    struct Gadget * MakeGadgets(struct Screen *,APTR ,ULONG,
             struct NewGadgets *, ULONG *, ULONG *,
             struct Gadget **, struct TextAttr *, float, float);

FUNCTION
```

```
    This function will create Gadgets from an array of
    Gadtools NewGadgets, scaling the gadgets to the size
    specified.

INPUTS
    Scr - This is the screen that the window containing these
             gadgets are destine for.  This is only used to get
             the WBorX values used to offset the gadgets from
             the window border.  *Set this to NULL if the gadgets
             are destine for a GimmeZeroZero window*.
    VisualInfo - VisualInfo pointer from gadtools/GetVisualInfo().
    NumGadgets - the number of gadgets to process.
    NewGads - array of struct NewGadget.
    NewGadTags - array of ULONGS to be processed as TagItems.
          All the gadget's tags are in this single array.
          (In the same fashion as GadToolsBox)
          Individual tag arrays are NULL(TAG_DONE) terminated,
          And you may use TAG_MORE for additional Tags per gadget,
          but the array still must be NULL terminated.
    NewGadKinds - array of ULONGs, (BUTTON_KIND etc.)
    Gadgets - (struct Gadget **) Pointer to an array of
          Gadget pointers.  The Gadgets created by this
          function can be referenced by the ng_GadgetID of
          the source NewGadget.
    TextAttr - (struct TextAttr *) This TextAttr to used for
          gadgets whose ng_TextAttr is NULL.
    XScale - the x factor to scale the gadgets.
    YScale - the y factor to scale the gadgets.

RESULT
    returns a pointer to the context from CreateContext()
    or NULL on failure.

EXAMPLE

NOTES
    requires diskfont, gadtools & utility libraries to
    be open.

    This function has a special flag for NewGadget->ng_Flags,
    NG_FITLABEL.  This flag tells the function to adjust the
    ng_LeftEdge and/or Width if the PLACETEXT_LEFT or RIGHT
    flags are set, and the ng_TopEdge and/or Height if the
    PLACETEXT_ABOVE or BELOW flags are set, so that the
    ng_GadgetText will fit inside the area specified for the
    gadget.  You must also specify one of the PLACETEXT_?
    values or this will not work.

    For GimmeZeroZero windows set Scr to NULL.

    This function will not modify any of the arrays
    (ie. NewGads, NewGadTags), this way if you have
    an interface that opens & closes multiple times
    (like commodities) you will not have to reinitialize
    the arrays.

    Every ng_GadgetID should be unique and none may have a
```

```
    larger value than the number of elements in the array
    pointed to by Gadgets.

    GTLV_ShowSelected works differently with this function.
    specify -1l for the display only gadget, specify the
    gadget _id_ of the string gadget for the editable gadget
    (the string gadget must be created before the listview).
    Gadtools wants a NULL or poiner to a gadget for this value
    but this is taken care by this function.

    Gadgets created by this function can be freed with a
    call to FreeGadgets().

BUGS

SEE ALSO
    GetGUIScale()
```

1.50 extras.lib/OBSOLETEDrawBevelBoxes

```
NAME
    DrawBevelBoxes -- draw a series of scaled bevel boxws.

SYNOPSIS
    DrawBevelBoxes(Window, VisualInfo, BBoxes, NumBoxes,
          XScale, YScale);

    void DrawBevelBoxes(struct Window *, APTR,
          struct BevelBox *, LONG, float, float);

FUNCTION
    Draws a series of scaled boxes.

INPUTS
    Win - the Window to draw the bevel boxes in.
    VI - VisualInfo previously obtained by
         gadtools.library/GetVisualInfoA()
    BBoxes - pointer to an array of struct BevelBox.
    NumBoxes - the number of entries in the array.
    XScale -
    YScale -

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.51 extras.lib/PD_PackData

```
NAME
    PD_PackData -- Pack supplied data into a memory block.

SYNOPSIS
    packeddata = PD_PackData(Tags)

    struct PackedData *PD_PackData(Tag Tags, ... )

FUNCTION
    Packs supplied data into a memory block, suitable for writing
    to disk.

INPUTS
    Tags - a TagList.
        PD_Version - Adds a ULONG into the memory block.
              Sets the version of the data.

        PD_BYTE - Adds a BYTE into the memory block.
        PD_UBYTE - Adds a UBYTE into the memory block.

        PD_WORD - Adds a WORD into the memory block.
        PD_UWORD - Adds a UWORD into the memory block.

        PD_LONG - Adds a LONG into the memory block.
        PD_ULONG - Adds a ULONG into the memory block.

        PD_STRPTR - Adds a NULL terminated string into the memory block.
              NULL pointers are supported

        PD_APTRSize - Adds the ULONG into the memory block,
              Sets size for subsequent PD_APTRs.
        PD_APTR - Adds data into the memory block.  Size
              of data is set by previous PD_APTRSize.
              if NULL, PD_APtrSize bytes are still written but are
              unset.

        PD_BufferSize - Set the size of subsequent PD_Buffers
              This tag does not add data to buffer.

        PD_Buffer     - Adds size of buffer, ULONG, followed by
              data pointed to by ti_Data.
              See note for PD_UnpackData(()

        PD_MemoryFlags - Sets MemoryFlags for allocating


RESULT

EXAMPLE

NOTES
  Except for BYTEs, data is UWORD aligned, pads are inserted as needed.

  PackedData->pd_Data
  {
    ULONG privatelength
```

```
    {private bytes}
    user data
  }


BUGS

SEE ALSO
```

1.52 extras.lib/PD_UnpackData

```
NAME
    PD_UnpackData -- Unpack memory block.

SYNOPSIS
    success = PD_UnpackData(Tags)

    BOOL *PD_UnpackData(Tag Tags, ... )

FUNCTION
    Packs supplied data into a memory block, suitable for writing
    to disk.

INPUTS
    Tags - a TagList.
        For most tags, ti_Data is a storage pointer, if NULL the
        data is parsed but not stored.

        PD_Version   - (ULONG *) Reads the version of the data.

        PD_IfVersion - (ULONG) Once PD_Version has been read, you can
            this tag to stop data processing if PD_Version is lower than
            the value of this tag.

        PD_BYTE - (BYTE *) Read a BYTE
        PD_UBYTE -(UBYTE *)

        PD_WORD - (WORD *) Read a WORD
        PD_UWORD -(UWORD *)

        PD_LONG - (LONG *) Read a LONG
        PD_ULONG -(LONG *)

        PD_STRPTR - (STRPTR *) AllocVec()s Memory for new STRPTR
            if PD_FreeSTRPTR is set, existing STRPTR is FreeVec()ed.

        PD_APTRSize - (ULONG *) Reads the amount of bytes that subsequent
            PD_APTRs read from the packed data.

        PD_APTR - Reads data from the memory block.  Size
              of data is set by previous PD_APTRSize.
              if NULL, PD_APtrSize bytes are still written but are
              unset.

        PD_BufferSize - (ULONG) Set data size for subsequent PD_Struct
```

```
                reads. This tag does not read any data, you must specify
                the size of your Buffer.

            PD_Buffer - (APTR) Data is copied to existing memory, the amount
                of data copied will be the lesser of PD_BufferSize or the
                size stored in the packed data block.

            PD_MemoryFlags - Sets MemoryFlags for allocating


    RESULT

    EXAMPLE
        PD_UnpackData(pd,
            PD_Version,     &dataversion, // store version number
            PD_BYTE,        &byte1,
            PD_BYTE,        0,            // byte not stored.
            PD_STRPTR,      &string,

            PD_IfVersion,   1, /* the following will only be read id PD_Versio
n >= 1 */
            PD_APTRSize,    10,
            PD_APTR,        &memchunk1,
            PD_APTR,        &memchunk2,

            PD_IfVersion,   2, /* the following will only be read id PD_Versio
n >= 2 */
            PD_STRUCT(mystruct),

            TAG_DONE,       0))

    BUGS

    SEE ALSO
```

1.53 extras.lib/PhraseInStr

```
    NAME
        PhraseInStr -- Find a phrase or word in a string.

    SYNOPSIS
        str = PhraseInStr(InStr, Phrase)

        STRPTR PhraseInStr(STRPTR , Phrase)

    FUNCTION
        Locates a phrase or word in a string.

    INPUTS
        InStr - String to search in.
        Phrase to search for.

    RESULT
        returns pointer to phrase in InStr or NULL.
```

```
EXAMPLE

NOTES
    case insensitive.

BUGS

SEE ALSO
```

1.54 extras.lib/ProcessTagList

```
NAME
    ProcessTagList -- Macro to process a taglist

SYNOPSIS
    ProcessTagList(TagList, Tag, TState)

    TState=TagList;
    while(Tag=NextTagItem(&TState))

EXAMPLE
    void SomeFunc(struct TagItem *TagList)
    {
      struct TagItem *tag, *tstate;

      ProcessTagList(TagList,tag,tstate)
      {
        seitch(tag->ti_Tag)
        {
          case GA_Left:
            ...
            break;
          etc...
        }
      }
    }
```

1.55 extras.lib/RenderText

```
NAME
    RenderTextA -- Write text into a RastPort.
    RenderText -- varargs stub.

SYNOPSIS
    reserved = RenderTextA( RP, String, TagList)

    LONG RenderTextA( struct RastPort *, STRPTR,
                      struct TagItem *);

    reserved = RenderText( RP, String, Tags, ... )

    LONG RenderText( struct RastPort *, STRPTR,
```

```
                    Tag, ...);

FUNCTION
    Writes text into a Rastport, basically an interface
    to the graphics.library/Text() function.

INPUTS
    RP - rastport to write to.
    String - the string to write.
    TagList - an array of TagItems.
      RT_Baseline - baseline of the cursor.
                    default RastPort->cp_y
      RT_XPos     - horizontal position of the cursor.
                    default RastPort->cp_x
      RT_MaxWidth - maximum pixel length of text, excess
                    characters will be clipped.
      RT_Justification - RTJ_???  (default _LEFT)
      RT_TextFont - struct TextFont * from OpenFont()
      RT_Strlen   - number of characters in string.
      RT_TextLength - (ULONG *) Width in pixels of printed texted.


RESULT
    Number of characters drawn.

BUGS
    RenderText does not reset a RastPort's TextFont
    after using RT_TextFont.

SEE ALSO
    mlr_rendertext.image image class.
```

1.56 extras.lib/str_Strip

```
NAME
    str_Strip - Remove leading and trailing white spaces.

SYNOPSIS
    str_Strip(Str)

    void Strip(STRPTR);

FUNCTION
    This function removes leading and trailing whites-paces
    from a string.  White-spaces are determined by the
    IsWhiteSpace() function.

INPUTS
    Str - a null terminated string pointer.

RESULT
    none.

EXAMPLE
```

```
NOTES
    Modifies existing string memory.

BUGS
    Will not work on NNStr used by other some functions in
    the extras.lib.

SEE ALSO
    IsWhiteSpace().
```

1.57 extras.lib/StrIStr

```
NAME
    StrIStr -- search for a string in another string, case
               insenseitive.

SYNOPSIS
    StrIStr(InStr,SearchStr)

    STRPTR StrIStr(STRPTR InStr,STRPTR SearchStr);

FUNCTION
    Seach for a string inside another, case insensitive.

INPUTS
    InStr - the string to search in.
    SearchStr - the string to search for.

RESULT
    returns a pointer in InStr that matches SearchStr, or NULL
    if no match was found.
```

1.58 extras.lib/tag_AddTag

```
NAME
    tag_AddTag -- Add a tag toa taglist.

SYNOPSIS
    ok = tag_AddTag(TagList, Tag, Data)

    BOOL tag_AddTag(struct TagItem *, ULONG, ULONG);

FUNCTION
    Add a tag pair to a taglist created with tag_AllocTags()

INPUTS
    TagList - TagList created with tag_AllocTags()
    Tag - ti_Tag value
    Data - ti_Data value

RESULT
    non zero if the tag was added.
```

```
    failure can be due to under sized taglist.

EXAMPLE
    see tag_AllocTags()

NOTES
    Don't tag_AddTag TAG_IGNORE, TAG_DONE, TAG_MORE, TAG_SKIP.
    This function will only effect the specified TagList, and
    not any other lists referenced by TAG_MORE.
    This function overwrites existing same tags.

SEE ALSO
    tag_AllocTags()
```

1.59 extras.lib/tag_AddTags

```
NAME
    tag_AddTags -- Add a taglist to a taglist.

SYNOPSIS
    ok = tag_AddTags(TagList, Tag, Data)

    BOOL tag_AddTags(struct TagItem *, ULONG, ULONG);

FUNCTION
    Add a taglist to a taglist created with tag_AllocTags()

INPUTS
    TagList - TagList created with tag_AllocTags()
    NewTags - Tags to add to TagList

RESULT
    non zero if the tag was added.
    failure can be due to under sized taglist.

EXAMPLE
    see tag_AllocTags()

NOTES
    This function will only effect the specified TagList, and
    not any other lists referenced by TAG_MORE.
    This function overwrites existing same tags.

SEE ALSO
    tag_AllocTags()
```

1.60 extras.lib/tag_AllocTags

```
NAME
    tag_AllocTags -- Allocate blank Tag List

SYNOPSIS
```

```
    taglist = tag_AllocTags(TagCount)

    struct TagItem *tag_AllocTags(ULONG);

FUNCTION
    Allocate tag space for use with other tag_? functions.

INPUTS
    TagCount - Number of blank tags to allocate.

RESULT
    An empty tag space ending with TAG_DONE, or NULL.

EXAMPLE

NOTES

BUGS

SEE ALSO
```

1.61 extras.lib/tag_ClearNumTags

```
NAME
    tag_ClearTags -- Clear a TagList

SYNOPSIS
    void tag_ClearTags(TagList, TagCount)

    tag_ClearTags(struct TagItem *, ULONG);

FUNCTION
    Clears the TagList of all data.

INPUTS
    TagList - Allocated with tag_AllocTags()
    TagCount - Number of blank tags to allocate.

EXAMPLE
    see tag_AllocTags()

NOTES
    This function is called by tag_AllocTags(), so
    the taglist is cleared when allocated.
    This function will only effect the specified TagList, and
    not any other lists referenced by TAG_MORE.

BUGS

SEE ALSO
    see tag_AllocTags()
```

1.62 extras.lib/tag_ClearTags

```
NAME
    tag_ClearTags -- Clear a TagList

SYNOPSIS
    void tag_ClearTags(TagList)

    tag_ClearTags(struct TagItem *);

FUNCTION
    Clears the TagList of all data.

INPUTS
    TagList - Allocated with tag_AllocTags()

EXAMPLE
    see tag_AllocTags()

NOTES
    This function will only effect the specified TagList, and
    not any other lists referenced by TAG_MORE.

BUGS

SEE ALSO
    see tag_AllocTags()
```

1.63 extras.lib/tag_CountUserTags

```
NAME
    tag_CountUserTags -- number of user tags

SYNOPSIS
    ULONG tag_CountUserTags(TagList)

    tag_CountUserTags(struct TagItem *);

FUNCTION
    Count the number of user tags.

INPUTS
    TagList - Allocated with tag_AllocTags().
```

1.64 extras.lib/tag_FreeTags

```
NAME
    tag_FreeTags -- Clear a TagList

SYNOPSIS
    void tag_FreeTags(TagList)
```

```
    tag_FreeTags(struct TagItem *);

FUNCTION
    Frees the TagList.

INPUTS
    TagList - Allocated with tag_AllocTags().

EXAMPLE
    see tag_AllocTags()

SEE ALSO
    see tag_AllocTags()
```

1.65 extras.lib/tag_RemTag

```
NAME
    tag_RemTag -- Removea a tag to a taglist.

SYNOPSIS
    ok = tag_RemTag(TagList, Tag)

    BOOL tag_RemTag(struct TagItem *, ULONG);

FUNCTION
    Find and remove a tag from a taglist.

INPUTS
    TagList - TagList created with tag_AllocTags()
    Tag - ti_Tag value

RESULT
    non zero if the tag was found and removed.

EXAMPLE
    see tag_AllocTags()

NOTES
    Don't tag_AddTag TAG_IGNORE, TAG_DONE, TAG_MORE, TAG_SKIP.
    This function will only effect the specified TagList, and
    not any other lists referenced by TAG_MORE.

SEE ALSO
    tag_AllocTags()
```

1.66 extras.lib/tag_TagDone

```
NAME
    tag_TagDone -- End the TagList with TagDone

SYNOPSIS
    void tag_FreeTags(TagList)
```

```
    tag_FreeTags(struct TagItem *);

FUNCTION
    Ends the taglist with TAG_MORE and link the list to MoreTags

INPUTS
    TagList - Allocated with tag_AllocTags().

EXAMPLE
    see tag_AllocTags()

SEE ALSO
    see tag_AllocTags()
```

1.67 extras.lib/tag_TagMore

```
NAME
    tag_TagMore -- End the TagList with TagMore

SYNOPSIS
    void tag_FreeTags(TagList, MoreTags)

    tag_FreeTags(struct TagItem *, struct TagItem *);

FUNCTION
    Ends the taglist with TAG_MORE and link the list to MoreTags

INPUTS
    TagList - Allocated with tag_AllocTags().
    MoreTags - Tags to link

EXAMPLE
    see tag_AllocTags()

SEE ALSO
    see tag_AllocTags()
```

1.68 extras.lib/thread_EndThread

```
NAME
    thread_EndThread -- end a thread.

SYNOPSIS




FUNCTION
```

```
INPUTS

RESULT

EXAMPLE

NOTES
    Note, this function waits for a reply.

BUGS

SEE ALSO
```

1.69 extras.lib/thread_PutTMsg

```
NAME
    thread_PutTMsg --

SYNOPSIS
    success=thread_PutTMsg(Thread, Msg)

    BOOL thread_PutTMsg(struct Thread *, struct ThreadMessage *)

FUNCTION
    Send a message to the thread, this function does not
    wait for a reply.  You must supply a reply port for
    your message.

INPUTS
    Thread - to send message to
    Msg - Message to send.

RESULT
    Non-zero if message successfully sent.

NOTES
    Note, this function does not wait for a reply.
```

1.70 extras.lib/thread_PutTMsg_Sync

```
NAME
    thread_PutTMsg_Sync -- Send a ThreadMessage to a thread.

SYNOPSIS
    success=thread_PutTMsg_Sync(Thread, Msg)

    BOOL thread_PutTMsg_Sync(Thread, struct ThreadMessage);

FUNCTION
    Send a message to the thread, this function will
    wait for a reply.  The messages reply port will be changed.
```

```
   INPUTS
       Thread - to send message to
       Msg - Message to send.

   RESULT
       Non-zero if message successfully sent.

   NOTES
       Note, this function waits for a reply.
       replyport is changed

   BUGS

   SEE ALSO
```

1.71 extras.lib/thread_PutTMsg_TagList

```
   NAME
       thread_PutTMsg_TagList -- Send a TagListTMsg to a thread (varargs)

   SYNOPSIS
       RetVal = thread_PutTMsg_TagList(Thread, Command, Tags ... )

       ULONG thread_PutTMsg_TagList(struct Thread, ULONG, Tag, ...);

   FUNCTION
       Sends a message to the task, using the TMsg_TagList structure.
       Waits for a reply, then returns tm_RetVal.

   INPUTS
       Thread - (struct Thread *)
       Command - Command ID.
       Tag -

   RESULT
       returns zero on failure, otherwise returns value of TMsg_TagList.tm_Re
tVal.

   NOTES
       Note, this function waits for a reply.
```

1.72 extras.lib/thread_StartThread

```
   NAME
       thread_StartThread -- Create a thread.

   SYNOPSIS
       Thread thread_StartThread(Tags)

       struct Thread *thread_StartThread(Tag, ... );
```

```
FUNCTION
    Creates and starts a new thread (Process).

INPUTS
    Tags - TagList (on stack)
      TA_Name - Name of thread, defaults to "Thread".
      TA_Stack - Stacksize, default 8192.
      TA_Priority - default -1
      TA_MsgHandler - Function to handle thread messages.
      TA_UserData - (APTR)
      TA_A6 - (struct Library *)

RESULT
    Pointer to the newly created Thread, or NULL on failure.
    The Thread stucture is ReadOnly, except for
      t_Node, UserData and ThreadData

NOTES
    It's suggested that ThreadData be used store local data
    for the Thread.

BUGS
    1.3 - Made functional, some timing issues caused crashes.

SEE ALSO
```

1.73 extras.lib/UpdateProgressMeterA

```
NAME
    UpdateProgressMeterA -- Change ProgressMeter attributes.
    UpdateProgressMeter -- varargs stub.

SYNOPSIS
    numProcessed UpdateProgressMeterA(PM,TagList)

    LONG UpdateProgressMeterA(ProgressMeter ,struct TagItem *);

    numProcessed UpdateProgressMeter(PM,FirstTag)

    LONG UpdateProgressMeter(ProgressMeter , Tag, ...);

FUNCTION
    Updates a ProgressMeter's attributes  and refreshes
    it as neccessary.

INPUTS
    PM - Pointer to an existing ProgressMeter or NULL.
    TagList - TagList of attributes to change or NULL.
        Only these four tags are processed.
          PM_QueryCancel
          PM_MeterValue
          PM_LowValue
          PM_HighValue

RESULT
```

```
    returns the number of tags processed.

EXAMPLE

NOTES
  requires diskfont, exec, gadtools, graphics, intuition & utility
  libraries to be open.

BUGS

SEE ALSO
  AllocProgressMeterA(), FreeProgressMeter()
```

1.74 Macro/nns_ProcessNNStr

```
NAME
    nns_ProcessNNStr(NNStr, Str)

SYNOPSIS
    nns_ProcessNNStr(NNStr, Str)

FUNCTION

INPUTS

RESULT

EXAMPLE
    STRPTR NNStr, str;

    ProcessNNStr(NNStr,str)
    {
      printf("%s\n",str);
    }

NOTES

BUGS

SEE ALSO
```