english_gtdrag

COLLABORATORS			
	TITLE : english_gtdrag		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 14, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

english_gtdrag

1.1 gtdrag - Documentation

```
                gtdrag - Version 3, Revision 2, 10.6.1999

This package is copyrighted 1996-99 by pinc Software,
                Axel Dörfler
                .
All rights reserved.


                Copyright notice

                Introduction

                System requirements

                Functions

                BOOPSI Methods

                Author

                Credits

                Release Notes

                Known Bugs

                To do
                This is a final release of the package. This means that it
implements most yet planned features and that the API will
stay compatible with upcoming releases.
```

1.2 gtdrag - Copyright notice

```
                The package is ©1996-99 by pinc Software,
```

1.3 gtdrag - The Author

You can reach me under the following address:

Snail-mail: Axel Dörfler
Heerstraße 53
49492 Westerkappeln
Germany

e-Mail: axeld@bigfoot.de

If you find a bug or have any suggestions what to implement in
gtdrag or if you want to become a beta-tester of this library,
please write me!

http://www-lehre.informatik.uni-osnabrueck.de/~adoerfle/

or (sooner or later):

http://www.pinc-software.de/

The gtdrag.library is compiled using the SAS/C-Compiler 6.59 on
an A4000/040 with OS3.0 (V39).

1.4 gtdrag - Credits

Thanks to:
- Alexander Bartz (stimpleton@bigfoot.de) for bug report and support.

- Guido Mersmann for bug reports and ideas.

- Daniel Rost (d96rost@ios.chalmers.se) for the creation of the
  E-modules and include files for gtdrag and the translation of
  the example program.

1.5 gtdrag - Introduction

The gtdrag shared library provides the possibility of the
drag&drop-feature for gadtools- and boopsi-applications.
It is thought to close the gap between now and an upcoming
version of AmigaOS with this features included. If this
will ever be included in an AmigaOS.

It is definitely not the best you can imagine, but it works
good and is easy to use.
You drag an entry of listview when you select it and move the
mouse horizontal more than 10 pixels. On other gadget types
except for custom gadgets (BOOPSI), you have to move >3 pixels
to start dragging. Perhaps I'll implement a special "dragkey"
like "alt" or whatsoever.

The drag&drop feature is not limited to a single window or
application, every gadget you add or which supports drag&drop,
is included.
But of course, it is limited to one screen at the same time.

The gtdrag library extends the Gadget Toolkit but also includes
all boopsi gadgets in a smart and compatible way. Custom gadgets
which support drag&drop are recognised automatically.

Furthermore, it provides a set of rendering callback hooks for
listviews which are very powerful (e.g. a tree-view).

1.6 gtdrag - System requirements

gtdrag should work on any Amiga with OS 3.0+ (V39 and above).
It is not possible to use it under OS 2.04.

But you can also build an application which runs under 2.04
and an installed gtdrag.library - in this case gtdrag works
as a gate to gadtools without the dragging capabilities.

This is also the case if you start an application which do
not support the new version 3 of this library.

1.7 gtdrag - Function Overview

TABLE OF CONTENTS

gtdrag.library/GTD_AddGadgetA

gtdrag.library/GTD_AddWindowA

gtdrag.library/GTD_BeginDrag

gtdrag.library/GTD_FilterIMsg

gtdrag.library/GTD_GetAttr

gtdrag.library/GTD_GetHook

gtdrag.library/GTD_GetIMsg

gtdrag.library/GTD_GetString

gtdrag.library/GTD_HandleInput

gtdrag.library/GTD_PostFilterIMsg

gtdrag.library/GTD_PrepareDrag

gtdrag.library/GTD_RemoveApp

gtdrag.library/GTD_RemoveGadget

gtdrag.library/GTD_RemoveGadgets

gtdrag.library/GTD_RemoveWindow

gtdrag.library/GTD_ReplyIMsg

gtdrag.library/GTD_SetAttrsA

gtdrag.library/GTD_StopDrag
Tree support functions:

gtdrag.library/AddTreeNode

gtdrag.library/CloseTreeNode

gtdrag.library/FindListSpecial

gtdrag.library/FindTreePath

gtdrag.library/FindTreeSpecial

gtdrag.library/FreeTreeList

gtdrag.library/FreeTreeNodes

gtdrag.library/GetTreeContainer

gtdrag.library/GetTreePath

```
            gtdrag.library/InitTreeList

            gtdrag.library/OpenTreeNode

            gtdrag.library/ToggleTree

            gtdrag.library/ToggleTreeNode
```

1.8 gtdrag.library/GTD_AddAppA

```
                NAME
    GTD_AddAppA -- inits all app. dragging context data
    GTD_AddApp  -- Varargs stub for GTD_GetAppA()

SYNOPSIS
    result = GTD_AddAppA(name,taglist)
    d0                   a0   a1

    int GTD_AddAppA(STRPTR,struct TagItem *);


    result = GTD_AddApp(name,firsttag,...)

    int GTD_AddApp(STRPTR,Tag,...);

FUNCTION
    This function allocates and sets all needed data.
    You must call it before any other action with this library.
    Otherwise your calls are ignored.
    Every name must be unique - this enables the possibility to
    support one or more special applications.

TAGS
    GTDA_InternalOnly (BOOL) - if you don't want to react on
      dropped objects of other applications, you should set this
      flag to TRUE. Defaults to FALSE.
      This does not affect the possibility of drag one of your
      objects to another application.

    GTDA_NewStyle (BOOL) - you must set this tag if
      you want gtdrag to work. (V3)

RESULT
    result - TRUE for success.

SEE ALSO

            GTD_RemoveApp()
            ,
            GTD_GetIMsg()
```

1.9 gtdrag.library/GTD_AddGadgetA

```
                NAME
    GTD_AddGadgetA -- adds a drag&drop gadget to the internal list
    GTD_AddGadget  -- Varargs stub for GTD_AddGadgetA()

SYNOPSIS
    GTD_AddGadgetA(type,gad,win,taglist)
                   d0   a0  a1  a2

    void GTD_AddGadgetA(int,struct Gadget *,struct Window *,struct TagItem *);


    GTD_AddGadget(type,gad,win,firsttag,...)

    void GTD_AddGadget(int,struct Gadget *,struct Window *,Tag,...);

FUNCTION
    This function adds the chosen gadget to the internal list.
    You must call this function, if you want a gadget to have
    the drag&drop-feature.
    A specially supported type is the LISTVIEW_KIND. The TREEVIEW_KIND
    and the IMAGEVIEW_KIND are similar but support special features
    of the corresponding render hook.

    You may also add boopsi gadgets to your gadget list. Doing so
    will ensure that gtdrag knows that such a gadget is attached
    to your application. This is not necessary, if the custom gadget
    is in the same window as a registered (to gtdrag) gadget or if
    the window is registered itself.
    If that is not the case, the gadget's application would be
    "intuition" - which is not always the desired effect.

    You should not set any tags for custom gadgets. They should
    handle these for themselves. You should read the appropriate
    boopsi gadget documentation for possibly allowed tags.
    GTDA_InternalOnly may be such a candidad.

TAGS
    see
            GTD_SetAttrsA()
                INPUTS
    type - type of the gadget (e.g. LISTVIEW_KIND)
    gad - pointer to the gadget (from CreateGadget()).
    win - pointer to the window of the gadget

NOTES
    If the gadget type is not (LIST|TREE|IMAGE)VIEW_KIND, GTDA_Object
    and GTDA_Images or GTDA_RenderHook must be set, or DGF_NODRAG will
    be set.
    Furthermore, you have to set the GA_Immediate-attribute to
    TRUE if want to have a draggable gadget.
    The function should only be used with GTDA_NoDrag when using
    other gadget types than LISTVIEW_KIND or BUTTON_KIND. Otherwise
    it could make problems (check it out).
```

```
    You MUST set the GadgetID field of your gadgets!

SEE ALSO

            GTD_RemoveGadget()
            ,
            GTD_RemoveApp()
            ,
            GTD_SetAttrsA()
```

1.10 gtdrag.library/GTD_AddWindowA

```
                NAME
    GTD_AddWindowA -- adds drag&drop notify to a window
    GTD_AddWindow  -- Varargs stub for GTD_AddWindowA()

SYNOPSIS
    GTD_AddWindowA(win,taglist)
                   a0  a1

    void GTD_AddWindowA(struct Window *,struct TagItem *);


    GTD_AddWindow(win,firsttag,...)

    void GTD_AddWindow(struct Window *,Tag,...);

FUNCTION
    This function adds the chosen window to the internal list.
    You must call this function, if you want to receive drag&
    drop messages from a window.

    If a custom gadget is in this window, the custom gadget's
    application will set to the one of the window.

TAGS
    GTDA_AcceptTypes (ULONG) - if the drag is an application-internal drag
      and the type value of the dropped object and (bit-wise) this value
      are TRUE, the drag will be accepted.
      Defaults to 0xffffffff (all internal types).

INPUTS
    win - pointer to the window

NOTE
    You only receive a DMT_WINDOW message from GTD_GetDragMsg() if
    the object wasn't dropped over a gadget which supports drag&drop.

SEE ALSO

            GTD_RemoveWindow()
            ,
            GTD_RemoveApp()
```

1.11 gtdrag.library/GTD_BeginDrag

```
                NAME
    GTD_BeginDrag -- starts the drag of an object

SYNOPSIS
    success = GTD_BeginDrag(gad,gpi)
    d0                      a0  a1

    BOOL GTD_BeginDrag(struct Gadget *,struct gpInput *);

FUNCTION
    This function starts the drag after the
            GTD_PrepareDrag()
                    call and the successing object specification via
            GTD_SetAttrsA()
            .

    If you call this functions, you also have to call GTD_StopDrag
    in the GM_GOINACTIVE method to break the drag cleanly.

INPUTS
    gad - pointer to the custom gadget
    gpi - the gpInput method structure

RESULT
    returns TRUE if the drag could be started, or FALSE if not.

SEE ALSO

            GTD_PrepareDrag()
            ,
            GTD_HandleInput()
            ,
            GTD_SetAttrsA()
            ,
            GTD_StopDrag
```

1.12 gtdrag.library/GTD_FilterIMsg

```
                NAME
    GTD_FilterIMsg -- filter an IntuiMessage through gtdrag & gadtools.

SYNOPSIS
    modmsg = GTD_FilterIMsg(imsg)
    d0                      a0

    struct IntuiMessage *GTD_FilterIMsg(imsg);
```

```
FUNCTION
    see gadtools/GT_FilterIMsg().

INPUT
    imsg - an IntuiMessage you got from GetMsg().

RESULT
    modmsg - pointer to the modified IntuiMessage

SEE ALSO

            GTD_PostFilterIMsg()
```

1.13 gtdrag.library/GTD_GetIMsg

```
                NAME
    GTD_GetIMsg -- get an IntuiMessage, with all necessary processing

SYNOPSIS
    msg = GTD_GetIMsg(port)
    d0                a0

    struct IntuiMessage *GTD_GetIMsg(iport);

FUNCTION
    Gets an IntuiMessage and does all the drag processing including
    a call to the aquivalent gadtools functions.

INPUT
    port - a pointer to the msg port

RESULT
    msg - pointer to the IntuiMessage

SEE ALSO

            GTD_ReplyIMsg()
```

1.14 gtdrag.library/GTD_HandleInput

```
                NAME
    GTD_HandleInput -- handle the input events during the drag

SYNOPSIS
    retval = GTD_HandleInput(gad,gpi)
    d0                       a0  a1

    ULONG GTD_HandleInput(struct Gadget *,struct gpInput *);

FUNCTION
```

```
    This function is called at the beginning of a GM_HANDLEINPUT
    method. You must call this function if you want gtdrag to
    work correctly during a drag.
    During normal input processing, it returns GMR_HANDLEYOURSELF
    which means that you can handle the method as if gtdrag does
    not exist.

INPUTS
    gad - pointer to the custom gadget
    gpi - the gpInput method structure

RESULT
    The result value can be one of the normal return codes for a
    GM_HANDLEINPUT method plus GMR_HANDLEYOURSELF if you need to
    handle the input on yourself.
    During the drag it returns GMR_MEACTIVE and a GMR_NOREUSE
    afterwards. You may catch these, if you need a special behaviour
    for your gadget, e.g. that it should be active even after a
    drag.

SEE ALSO

            GTD_PrepareDrag()
            ,
            GTD_BeginDrag()
```

1.15 gtdrag.library/GTD_PostFilterIMsg

```
                NAME
    GTD_PostFilterIMsg -- return filtered IntuiMessage from
                          GTD_FilterIMsg().

SYNOPSIS
    msg = GTD_PostFilterIMsg(modimsg)
    d0                       a0

    struct IntuiMessage *GTD_FilterIMsg(imsg);

FUNCTION
    see gadtools/GT_FilterIMsg().

INPUT
    modimsg - an IntuiMessage you got from GTD_FilterIMsg().

RESULT
    msg - pointer to the unmodified IntuiMessage

SEE ALSO

            GTD_FilterIMsg()
```

1.16 gtdrag.library/GTD_PrepareDrag

```
                NAME
    GTD_PrepareDrag -- prepares the drag of an object

SYNOPSIS
    success = GTD_PrepareDrag(gad,gpi)
    d0                        a0  a1

    BOOL GTD_PrepareDrag(struct Gadget *,struct gpInput *);

FUNCTION
    You call this function within the GM_HANDLEINPUT method.
    gtdrag will prepare it's internal data structures for the
    beginning of a drag.
    Furthermore, it tests, if there are any reasons for not
    starting the drag - this should only occur in very rare
    cases, e.g. if the left mouse button is not pressed.

    You may also check for qualifiers, if you need an extra
    key to be pressed to allow a drag, you should use ALT in
    this case.

    Between this call and
            GTD_BeginDrag()
            , you have to specify
    the object you want to be dragged and the attributes of your
    gadget via
            GTD_SetAttrsA()
            .

INPUTS
    gad - pointer to the custom gadget
    gpi - the gpInput method structure

RESULT
    returns TRUE if the drag could be started now, or FALSE if not.

SEE ALSO

            GTD_BeginDrag()
            ,
            GTD_HandleInput()
            ,
            GTD_SetAttrsA()
```

1.17 gtdrag.library/GTD_RemoveApp

```
                NAME
    GTD_RemoveApp -- frees the dragging context data

SYNOPSIS
    GTD_RemoveApp()
```

```
    void GTD_RemoveApp(void);

FUNCTION
    Frees all the memory allocated by GTD_AddApp(), GTD_AddGadget() and
    GTD_AddWindow() for an application.
    You should not call it without the GTD_AddApp() call before.

SEE ALSO

            GTD_AddAppA()
            ,
            GTD_AddGadgetA()
            ,
            GTD_AddWindowA()
```

1.18 gtdrag.library/GTD_RemoveGadget

```
                NAME
    GTD_RemoveGadget -- removes a drag&drop gadget

SYNOPSIS
    GTD_RemoveGadget(gad)
                     a0

    void GTD_RemoveGadget(struct Gadget *);

FUNCTION
    This function removes the drag&drop feature from a gadget
    which was added to it with GTD_AddGadget().

INPUTS
    gad - pointer to the gadget (e.g. from CreateGadget()).

SEE ALSO

            GTD_AddGadgetA()
            ,
            GTD_RemoveApp()
            ,
            GTD_RemoveGadgets()
```

1.19 gtdrag.library/GTD_RemoveGadgets

```
                NAME
    GTD_RemoveGadgets -- removes all drag&drop gadgets from a window  (V3)

SYNOPSIS
    GTD_RemoveGadgets(win)
                      a0
```

```
    void GTD_RemoveGadgets(struct Window *);

FUNCTION
    This function removes the drag&drop feature from all gadgets
    of the specified window.

INPUTS
    win - pointer to the window

SEE ALSO

            GTD_AddGadgetA()
            ,
            GTD_RemoveApp()
            ,
            GTD_RemoveGadget()
```

1.20 gtdrag.library/GTD_RemoveWindow

```
                NAME
    GTD_RemoveWindow -- removes drag&drop notify from a window

SYNOPSIS
    GTD_RemoveWindow(win)
                     a0

    void GTD_RemoveWindow(struct Window *);

FUNCTION
    Removes drag&drop notfiy from a window which was added with
    GTD_AddWindow() before.

INPUTS
    win - pointer to the window

SEE ALSO

            GTD_AddWindowA()
            ,
            GTD_RemoveApp()
```

1.21 gtdrag.library/GTD_ReplyIMsg

```
                NAME
    GTD_ReplyIMsg -- replies an IntuiMessage

SYNOPSIS
    GTD_ReplyIMsg(msg)
                  a0
```

```
    void GTD_ReplyIMsg(struct IntuiMessage *);

FUNCTION
    Replies the msg got from GTD_GetIMsg(). It works like
    GT_ReplyIMsg() - but you must call GTD_GetIMsg() before!

INPUT
    msg - a pointer to the msg to be replied, may be NULL.

SEE ALSO

            GTD_GetIMsg()
```

1.22 gtdrag.library/GTD_SetAttrsA

```
                NAME
    GTD_SetAttrsA -- set gadget/window attributes  (V3)
    GTD_SetAttrs  -- Varargs stub for GTD_SetAttrsA()  (V3)

SYNOPSIS
    GTD_SetAttrsA(obj,ti)
                  a0  a1

    void GTD_SetAttrsA(APTR obj,struct TagItem *ti);


    GTD_SetAttrs(obj,tag1,...);

    void GTD_SetAttrs(APTR obj,ULONG tag1,...);

FUNCTION
    Set the attributes of registered gadgets and windows.

    For BOOPSI-class implementors: use this function between the
    GTD_Prepare/BeginDrag() calls to set all attributes needed
    for the drag operation.

TAGS
    GTDA_Images (BOOL) - if specified, only the image of an ImageNode
      will be dragged. The GTDA_RenderHook must not be set. Defaults
      to FALSE.

    GTDA_NoDrag (BOOL) - if specified, you cannot drag an item of this
      gadget; you are only able to drop them over it. Defaults to FALSE.

    GTDA_Same (BOOL) - for listviews and custom gadgets. If specified,
      source and target of a drag can be the same. Defaults to FALSE.

    GTDA_NoPosition (BOOL) - for listviews only. If specified, there is
      no position highlighting or scrolling. Defaults to FALSE.

    GTDA_NoScrolling (BOOL) - for listviews only. If specified, gtdrag
      will not scroll in the listview if you drag over the upper or
```

lower border of the listview. Defaults to FALSE;

GTDA_DropOverItems (BOOL) - for listviews only. If specified, the objects are not dropped between the existing items, they are dropped upon them.
The flag DMF_DROPOVER will be set in the DropMessage's dm_Flags field. Defaults to FALSE.

GTDA_DropBetweenItems (BOOL) - for listviews only. Works only in conjunction with the GTDA_DropOverItems tag.
If specified, both methods of object dropping are possible. Defaults to FALSE.

GTDA_TreeView (BOOL) - for listviews only. This enables the special tree view behaviour (drag&drop between items and over container). The flag DMF_DROPOVER will be set in the DropMessage's dm_Flags field, if an item was dropped over an item. Defaults to FALSE.

GTDA_ItemHeight (short) - specifies the height of a listview entry. Defaults to the screen's font height.

GTDA_RenderHook (struct Hook *) - if provided, the dragged object will be drawn with this listview-callback hook. Otherwise (assuming that there is no image to be drawn) there is only a ClipBlit() proceeded, which will probably not work in combination with simple refresh windows. You can set a special size with GTDA_Width & GTDA_Height. Defaults to NULL.

GTDA_Image (struct Image *) - if provided, the image will be used when the object is dragged.

GTDA_Object (APTR) - for non-listview gadgets: the object to be dragged. If a render hook is provided, the object must be of a type accepted by the render hook.
If no render hook is provided and no image is set, a (struct Image *) or ODT_IMAGE is assumed for the objects type.

GTDA_GroupID (ULONG) - the group ID of the object. This conforms to the datatypes group IDs such as GID_TEXT or GID_PICTURE. See datatypes/datatypes.h for other IDs. Defaults to NULL.

GTDA_Type (ULONG) - the rough format of the object. You should specify a type your object fits in, if you want to support drag&drop to other applications. See libraries/gtdrag.h for the possible values. Defaults to ODT_UNKNOWN (or ODT_IMAGE, if no render hook or image is specified for the object).

GTDA_InternalType (ULONG) - this should be a bit mask which will be tested on internal drag&drops (with GTDA_AcceptTypes). Defaults to 0xffffffff.

GTDA_ObjectDescription (struct ObjectDescription *) - the contents of the structure are copied to the internal buffer. If you specify a NULL pointer, all contents of the internal ObjectDescription are set to NULL.

GTDA_ObjectFunc (APTR) - this function will be called whenever a

```
      drag begins. You can set object related attributes in this
      function.
      The function called must conform to this prototype:

      void func(struct Window *,struct Gadget *,struct ObjectDescription *, ↩
         LONG);
             a0              a1              a2                          d0

      The argument passed in d0 corresponds to the dm_SourceEntry field
      of a DropMessage. The only tags you are allowed to use are:
        GTDA_Object, GTDA_GroupID, GTDA_Type, GTDA_InternalType,
        GTDA_ObjectDescription, GTDA_SourceEntry.
      All other tags may produce unpredictable results.

    GTDA_AcceptTypes (ULONG) - if the drag is an application-internal drag
      and the type value of the dropped object and (bit-wise) this value
      are TRUE, the drag will be accepted.
      Defaults to 0xffffffff (all internal types).

    GTDA_AcceptFunc (APTR) - this function will be called, whenever
      the GTDA_AcceptTypes can not decide if the object should be
      accepted or not.
      The function called must conform to this prototype:

      BOOL func(struct Window *,struct Gadget *,struct ObjectDescription *);
             a0              a1              a2

      It returns TRUE, if it accepts the object or FALSE if not.
      If this tag is set to a window, the Gadget pointer (a1) is NULL.
      Defaults to NULL.

    GTDA_Width, GTDA_Height (short) - set the size of the image to
      be dragged.

    GTDA_SourceEntry (LONG) - for non-listviews only. This flag directly
      influences the dm_SourceEntry field of a DropMessage structure if
      your gadget is the source gadget. Defaults to NULL.

    GTDA_InternalOnly (BOOL) - gadget will not accept any application
      external dragged objects. Custom gadget implementors should not
      use this flag since it's not sure, to which application they
      belong. Defaults to FALSE.  (V3.2)

INPUT
    obj - a pointer to a registered gadget or window, may be NULL.

SEE ALSO

            GTD_AddGadgetA()
            ,
            GTD_GetAttr()
```

1.23 gtdrag.library/GTD_GetAttr

```
                NAME
    GTD_GetAttr -- get gadget/window attribute  (V3)

SYNOPSIS
    result = GTD_GetAttr(obj,tag,storage);
    d0                   a0  d0  a1

    BOOL GTD_GetAttr(APTR obj,ULONG tag,ULONG *storage);

FUNCTION
    Gets an attribute of a registered gadget/window.

TAGS
    GTDA_Image (struct Image *) - returns the image which would
      be used for a drag.

    GTDA_RenderHook (APTR) - returns a pointer to the current
      render hook used to display the object.

    GTDA_Width (WORD) - returns the current width of the object
      which would be dragged. May be 0.

    GTDA_Height (WORD) - returns the current height of the object
      which would be dragged. May be 0.

    GTDA_Object (APTR) - the object of the gadget's ObjectDescription.

    GTDA_GroupID (ULONG) - the group ID of the object.

    GTDA_Type (ULONG) - the rough format of the object.

    GTDA_InternalType (ULONG) - the type of the object used for internal
      drag&drop.

    GTDA_ObjectDescription (struct ObjectDescription *) - the whole
      ObjectDescription structure from the gadget. This structure is
      read only.

INPUT
    obj - a pointer to a registered gadget or window. It is safe
      to pass a NULL pointer.

RESULT
    result - TRUE if the tag data could be stored, or FALSE for failure.

SEE ALSO

            GTD_AddGadgetA()
            ,
            GTD_SetAttrsA()
```

1.24 gtdrag.library/GTD_GetHook

```
NAME
    GTD_GetHook -- gets one of the provided callback hooks  (V3)

SYNOPSIS
    hook = GTD_GetHook(type);
    d0                 d0

    void GTD_GetHook(ULONG type);

FUNCTION
    Lets an application access useful callback hooks for listview
    rendering and IFF-streaming.
    Currently, the following are provided:

    GTDH_IMAGE - renders a listview which contains ImageNodes.

    GTDH_TREE - renders a listview which contains TreeNodes. It can
      be used very simple in conjunction with the tree support functions.
      hook->h_Data must be the pen which will be used to render the
      line structure.

    GTDH_IFFSTREAM - simplifies reading and writing data from and to
      iff streams. hook->h_Data must point to a IFFStreamHookData
      structure. If the buffer is used to write to, the buffer is
      created automatically and resized if necessary.
      If is_Pool is set, it uses pool functions for the buffer. The
      buffer is not freed upon exit.

    See libraries/gtdrag.h for more details.

INPUT
    type - one of the GTDH_xxx constants.

RESULT
    hook - a pointer to the struct Hook.

SEE ALSO
    GTD_SetAttrs(), libraries/gtdrag.h
```

1.25 gtdrag.library/GTD_GetString

```
NAME
    GTD_GetString -- gets a string out of a struct ObjectDescription  (V3)

SYNOPSIS
    string = GTD_GetString(od,buffer,len);
    d0                     a0 a1     d0

    STRPTR GTD_GetString(struct ObjectDescription *od,STRPTR buffer,ULONG len) ↩
       ;

FUNCTION
    This function provides a simple method to get a string out of an
    struct ObjectDescription which is part of a struct DropMessage.
```

```
    It uses the od_Type field to scan the object for a descriptive
    string. If no string could be read, it returns NULL. Otherwise
    the string buffer will be returned.

INPUT
    od     - pointer to a struct ObjectDescription
    buffer - string buffer large enough to hold the result
    len    - length of the string buffer

RESULT
    string - the string buffer or NULL for failure
```

1.26 gtdrag.library/GTD_StopDrag

```
                NAME
    GTD_StopDrag -- stops/interrupts the drag of an object

SYNOPSIS
    GTD_StopDrag(gad)
                 a0

    void GTD_StopDrag(struct Gadget *);

FUNCTION
    If you start a drag on you own via GTD_BeginDrag()/GTD_HandleInput()
    you have to call this function in the GM_GOINACTIVE method.

    You can break a drag cleanly with this function if necessary or
    if intuition interrupts the activity of your gadget without asking.

    It is safe to call this function even if you have not started
    a drag, so you don't have to check for this.

INPUTS
    gad - pointer to the custom gadget
    gpi - the gpInput method structure

SEE ALSO

             GTD_BeginDrag()
             ,
             GTD_HandleInput()
             ,
             GTD_SetAttrsA()
```

1.27 gtdrag.library/AddTreeNode

```
                NAME
    AddTreeNode -- adds a TreeNode to a list
```

```
SYNOPSIS
    treenode = AddTreeNode(pool,tree,name,im,flags)
                           a0   a1   a2   a3 d0

    struct TreeNode *AddTreeNode(APTR,struct MinList *,STRPTR,struct Image *, ↩
       UWORD);

FUNCTION
    This function simplifies adding a new TreeNode to a tree list.
    You could also do this without the use of this function; there
    is no magic to add an item to a list.
    If TNF_SORT is set in the flags field, the TreeNode will be
    inserted in the list, sorted by name.

    The name and the image pointer must be valid while the the tree
    is displayed; they are simply referenced not copied.

    After the tree list is completed, you have to call InitTreeList()
    to create a view list out of it.
    FreeTreeNodes()

INPUTS
    pool  - a pointer to the memory pool
    tree  - the tree list (e.g. tl_Tree of a TreeList)
    name  - the name of the TreeNode; this should always be defined,
            although it is not checked here
    image - a pointer to an image, maybe NULL
    flags - one of the TNF_xxx flags

RETURN
    treenode - the TreeNode or NULL for failure

SEE ALSO

            InitTreeList()
```

1.28 gtdrag.library/CloseTreeNode

```
                NAME
    CloseTreeNode -- closes an open TreeNode

SYNOPSIS
    CloseTreeNode(main,tn)
                  a0   a1

    CloseTreeNode(struct MinList *,struct TreeNode *);

FUNCTION
    This function automatically removes all contents of a TreeNode
    and its sub-trees recursively.

    It does not affect the TNF_OPEN flag of a TreeNode!
    ToggleTreeNode()
```

```
INPUTS
    main - the view list
    tn   - the TreeNode to be closed

SEE ALSO

            OpenTreeNode()
```

1.29 gtdrag.library/FindListSpecial

```
                NAME
    FindListSpecial -- searches a list for the special pointer

SYNOPSIS
    treenode = FindListSpecial(list,special)
                               a0   a1

    struct TreeNode *FindListSpecial(struct MinList *,APTR);

FUNCTION
    This function is similar to FindTreeSpecial() but it does no
    recursive search. It simply scans the provided list for a
    TreeNode with this special pointer.

INPUTS
    list    - the list to be scanned
    special - the special pointer

RETURN
    treenode - the TreeNode found, or NULL, if there aren't any

SEE ALSO

            FindTreeSpecial()
```

1.30 gtdrag.library/FindTreePath

```
NAME
    FindTreePath -- finds a TreeNode with a path

SYNOPSIS
    treenode = FindTreePath(tree,path)
                            a0   a1

    struct TreeNode *FindTreePath(struct MinList *,STRPTR);

FUNCTION
    If you have the whole path of a TreeNode, this function
    finds the pointer to the TreeNode for you.
```

```
INPUTS
    tree - the tree list
    path - a path to the TreeNode, may be NULL

RETURN
    treenode - the TreeNode, if found any, or NULL
```

1.31 gtdrag.library/FindTreeSpecial

```
                NAME
    FindTreeSpecial -- searches a tree for a special pointer

SYNOPSIS
    treenode = FindTreeSpecial(tree,special)
                               a0   a1

    struct TreeNode *FindTreeSpecial(struct MinList *,APTR);

FUNCTION
    A TreeNode contains a special pointer like the UserData of a
    gadget or whatever.
    This function searches for the TreeNode with this special
    pointer, even NULL is accepted.

INPUTS
    tree    - the tree list
    special - the special pointer

RETURN
    treenode - the TreeNode found, or NULL, if there aren't any

SEE ALSO

            FindListSpecial()
```

1.32 gtdrag.library/FreeTreeList

```
                NAME
    FreeTreeList -- frees the TreeList contents

SYNOPSIS
    FreeTreeList(pool,tl)
                 a0   a1

    FreeTreeList(APTR,struct TreeList *);

FUNCTION
    This function makes use of the FreeTreeNodes() function but
    in addition to this, it resets the view list.

INPUTS
```

```
    the - pointer to the memory pool
    the - TreeList with view and tree list

SEE ALSO

            FreeTreeNodes()
```

1.33 gtdrag.library/FreeTreeNodes

```
                NAME
    FreeTreeNodes -- frees the tree nodes

SYNOPSIS
    FreeTreeNodes(pool,list)
                  a0   a1

    FreeTreeNodes(APTR,struct MinList *);

FUNCTION
    This function deletes the whole tree and its sub-trees recursively.
    Corresponding to AddTreeNode(), the name and the image are
    not freed.

INPUTS
    pool - the pointer to the memory pool the tree was allocated in
    list - the tree list

SEE ALSO

            AddTreeNode()
```

1.34 gtdrag.library/GetTreeContainer

```
NAME
    GetTreeContainer -- gets the container of a TreeNode

SYNOPSIS
    container = GetTreeContainer(tn)
                                 a0

    struct TreeNode *GetTreeContainer(struct TreeNode *);

FUNCTION
    This function searches for the container of a TreeNode; if the
    TreeNode is a root node, it returns NULL.

INPUTS
    tn - the TreeNode, may be NULL

RETURN
```

```
    container - the container TreeNode or NULL
```

1.35 gtdrag.library/GetTreePath

```
NAME
    GetTreePath -- finds the path of a TreeNode

SYNOPSIS
    path = GetTreePath(tn,buffer,len)
                       a0 a1     d0

    STRPTR GetTreePath(struct TreeNode *,STRPTR,LONG);

FUNCTION
    You can use this function to copy the TreeNode's path (without
    its name) to the provided buffer.

INPUTS
    tn     - the TreeNode
    buffer - a buffer large enough to hold the path
    len    - the length of the buffer

RETURN
    path - returns the buffer or NULL for failure
```

1.36 gtdrag.library/InitTreeList

```
NAME
    InitTreeList -- inits a tree list contents for viewing

SYNOPSIS
    InitTreeList(tl)
                 a0

    InitTreeList(struct TreeList *);

FUNCTION
    After the whole tree is built, this function inits the view
    list; the treeview callback hook can now display the tree.

    This is not a trivial thing to do, the usage of this function
    is recommended.

INPUTS
    tl - a pointer to the TreeList
```

1.37 gtdrag.library/OpenTreeNode

```
                NAME
    OpenTreeNode -- opens a closed TreeNode

SYNOPSIS
    count = OpenTreeNode(main,tn)
                         a0   a1

    LONG OpenTreeNode(struct MinList *,struct TreeNode *);

FUNCTION
    This function adds all contents of the TreeNode and its open
    sub-trees to the view list.

    It does not affect the TNF_OPEN flag of the TreeNode!
    ToggleTreeNode()

INPUTS
    main - the view list
    tn   - the TreeNode to be opened

RETURN
    count - the number of TreeNodes added to the view list

SEE ALSO

            CloseTreeNode()
```

1.38 gtdrag.library/ToggleTree

```
NAME
    ToggleTree -- simplifies the usage of ToggleTreeNode()

SYNOPSIS
    toggled = ToggleTree(gad,tn,msg)
                         a0  a1 a2

    BOOL ToggleTree(struct Gadget *,struct TreeNode *,struct IntuiMessage *);

FUNCTION
    You can use this function instead of ToggleTreeNode() if the user
    clicks on a TreeNode in your listview gadget.
    It automatically updates the gadget and takes care of changements
    to the position of the list due to additional nodes.
    The gadget must be registered via GTD_AddGadget() if it should do
    anything.
    GTD_ToggleTreeNode()

INPUTS
    gad - a pointer to a registered gadget
    tn  - the TreeNode the user clicked on
    msg - a pointer to the IntuiMessage which reports the click

RETURN
```

```
        toggled - returns TRUE if it really toggled the TreeNode

    BUGS
        In release 3.1 this function did perfectly nothing [if everything
        worked fine with GT_GetGadgetAttrs()].

    SEE ALSO
        GTD_AddGadget()
```

1.39 gtdrag.library/ToggleTreeNode

```
                    NAME
        ToggleTreeNode -- toggles a TreeNode between closes/open

    SYNOPSIS
        count = ToggleTreeNode(main,tn)
                               a0   a1

        LONG ToggleTreeNode(struct MinList *,struct TreeNode *);

    FUNCTION
        This function adds all contents of a closed TreeNode and its
        open sub-trees to the view list, or it removes the open TreeNode
        and its open sub-trees from the view list.
        It also takes care of the TNF_OPEN flag of the TreeNode.
        OpenTreeNode()

    INPUTS
        main - the view list
        tn   - the TreeNode to be opened

    RETURN
        count - the number of TreeNodes added to the view list

    SEE ALSO

                CloseTreeNode()
```

1.40 gtdrag - BOOPSI Methods

```
                With version 3 gtdrag introduces a set of new methods which
will be sent to gadgets which are potential targets of an
object drop:


                 GM_OBJECTDRAG

                 GM_RENDERDRAG

                 GM_OBJECTDROP
```

1.41 gtdrag - method description: GM_OBJECTDRAG

```
GM_OBJECTDRAG - this method is invoked whenever an object is
  dragged over your gadget and it is not dropped yet (during the drag).

  struct gpObjectDrag
  {
    ULONG  MethodID;
    struct ObjectDescription *gpod_Object;
    struct Gadget *gpod_Source;      /* pointer to the source gadget */
    struct
    {
      WORD X,Y;                      /* mouse coordinates */
    }      gpod_Mouse;
  };

  You can analyse the provided Object Description, the source gadget
  and the mouse coordinates, which are relative to the upper left
  corner of your gadget, to accept (GMR_ACCEPTOBJECT) or reject
  (GMR_REJECTOBJECT) the object.

  If you want to update the gadget imagery you can return an or'ed
  GMR_UPDATE - at the first invokation of GM_OBJECTDRAG, GM_RENDERDRAG
  is invoked automatically if you return GMR_ACCEPTOBJECT.

  If you do not want to get this message more than one time, e.g.
  when you don't want to be notified on mouse moves over your gadget,
  you can return an or'ed GMR_FINAL. This causes gtdrag to store
  you return code for the whole drag of this special object.

  NOTE: The standard application for a custom gadget is "intuition".
        An application have the possibility to change this via
        GTDA_AddGadgetA() or GTDA_AddWindowA().
        You should take this into account.
```

1.42 gtdrag - method description: GM_RENDERDRAG

```
GM_RENDERDRAG - this method is invoked whenever your
  gadget should reflect a dragged object over it with special
  highlighting.

  struct gpRenderDrag
  {
    ULONG  MethodID;
    struct GadgetInfo *gprd_GInfo;   /* gadget context */
    struct RastPort *gprd_RPort;     /* already for use */
    ULONG  gprd_Mode;                /* one of the GRENDER_xxx */
    struct
    {
      WORD X,Y;                      /* mouse coordinates */
    }      gprd_Mouse;
  };

  This structure is similar to gpRender but it introduces new
```

```
modes (like gpr_Redraw) and the mouse coordinates relative
to the upper left corner of the gadget.

The highlighting process consists of three different modes:

GRENDER_HIGHLIGHT - initial and progressive highlighting
GRENDER_INTERIM   - rendering necessary between two GRENDER_HIGHLIGHT
GRENDER_DELETE    - erases the highlighting

You have the possibility to let gtdrag render all of these
modes; return TRUE signals gtdrag that you process the
rendering on your own, return FALSE if you want gtdrag to
process the standard highlighting and refresh rendering.

So you may render only GRENDER_HIGHLIGHT yourself and let
the restauration of the gadget imagery one of gtdrags
topics.
```

1.43 gtdrag - method description: GM_OBJECTDROP

```
               GM_OBJECTDROP - this method is invoked if an object is
dropped over your gadget.

struct gpObjectDrop
{
  ULONG  MethodID;
  struct DropMessage *gpod_Message;
  ULONG  gpod_Qualifier;
};

If you want the application which created your gadget to handle
this message, you have write access to the dm_TargetEntry and
the dm_Flags fields - all other fields are READ-ONLY!
In this case you would return FALSE.

If you want to catch and "absorb" the message yourself, you
return TRUE - the application won't be notified and you can
process all needed changes.

Due to internal library design, this method is invoked before
the last
              GM_RENDERDRAG
               (with GRENDER_DELETE), so be careful
with changements to the gadget imagery - you have to prevent
gtdrag from delete the highlighting itself.

You can use the gpod_Qualifier to enable special behaviour
if e.g. the shift key is pressed while the object was dropped
over your gadget.
The use of the shift key is recommended if you need any extra
functions - try to avoid having more than 2 special keys,
because this will confuse the user.

NOTE: The standard application for a custom gadget is "intuition".
      An application have the possibility to change this via
```

```
        GTDA_AddGadgetA() or GTDA_AddWindowA().
        You should take this into account.
```

1.44 gtdrag - Release Notes

```
                gtdrag 1.1 (29.9.96)
   - first release.

gtdrag 1.4 (3.10.96)
   - big changes in structure and tag definitions!  (1.2)

   - other gadget types than LISTVIEW_KIND will be handled
     correctly (like BUTTON_KIND).  (1.2)

   - added
               GTD_RemoveGadget()
                and
               GTD_RemoveWindow()
                    for multi-window support.  (1.2)

   - checks now for the number of entries in a source-listview.
     (1.3)

   - the range of a mouse move until a drag happens can now be
     changed via GTDA_(LV)DragPixel.  (1.3)

   - the dragged icon now hides if you are near the source gadget
     to prevend graphical damage.  (1.4)

---------

gtdrag 1.8 (5.11.96)
   - created a shared library, the linker lib is not provided anymore
     but I can create one on request. This creation is part of big
     changes in the whole application interface. E. g. the GTD_Init()/
     Free() function-pair doesn't exist any longer.  (1.5)

   - implemented GTD_(Post)FilterIMsg()-functions.  (1.6)

   - checks now the number of entries in the target-listview.  (1.6)

   - makes the dragged object transparent if it is over a gadget
     which accepts drag&drop.  (1.6)

   - now centers the dragged object if it is an image.  (1.7)

   - fixed problems with more than one window open at a time.  (1.8)

   - fixed the GTD_(Post)FilterIMsg()-functions; they replied the
     same IntuiMessage two times.  (1.8)

gtdrag 2.1 (7.11.96)
   - implemented accept and type masks.  (1.9)
```

```
   - implemented GTDA_InternalOnly and GTDA_Same.  (1.9)

   - changed the handling of the DragMsgs internally.  (1.10)

   - bumped to V2 to let applications presuppose the extensions
     to the last release.  (2.1)

   - it was possible to get a drag message (DMT_UNKNOWN) if a drag
     was interrupted using the right mouse button.  (2.1)

gtdrag 2.2 (11.11.96)
   - if no DragGadget was defined, this could make problems.  (2.2)

   - fixed possible problems with the use of Wait() in applications.
     (2.2)

   - the GTDA_RenderHook got one pixel too much in width and height.
     (2.2)

gtdrag 2.7 (15.12.96)
   - the drag objects are now rendered via BitMaps, they are no BOBs
     any longer to reduce flickering on gfx-cards.  (2.3)

   - include the E includes/modules made by Daniel Rost.  (2.3)

   - fixed small bugs in the render-routine.  (2.3)

   - a little more highlighting is done.  (2.4)

   - the render-routines were a bit optimised and changed for use with
     CyberGraphX.  (2.5)

   - the message handling is completely changed, now it uses fake events
     to let gadtools stop processing messages while an object is dragged.
     (2.6)

   - small optimisations.  (2.6)

   - enhanced highlighting and listview-control. If you move an entry
     in a listview, you may get another dm_TargetEntry than in previous
     releases.  (2.6)

   - added scrolling capabilities to listview-drags.  (2.6)

   - some minor bug fixes.  (2.7)

gtdrag 2.10 (27.6.97)
   - a bug in the render-routine could cause a strange listview entry to
     appear when running under CyberGraphX with huge call-backs.  (2.8)

   - if the first IDCMP_MOUSEMOVE during a drag did not come from the
     source gadget, strange effects may happen.  (2.8)

   - no more "ghost" drags will occur when you release the mouse button
     while scrolling in a listview.  (2.9)

   - fixed a bug which could appear if the drag doesn't start over the
```

```
     gadget itself.  (2.10)

gtdrag 3.1 (8.6.99)
   - include a set of rendering callback-hooks.  (3.1)

   - add the GTD_RemoveGadgets() function which removes all the gadgets
     of a specific window.  (3.1)

   - implemented GTDA_ObjectFunc.  (3.1)

   - if a LISTVIEW_KIND didn't provide a render hook, the width of the
     ClipBlit() was one pixel too wide [reported by Guido Mersmann].
     (3.1)

   - slight API changes.  (3.1)

   - the TreeView's knob coordinates could be wrong in conjunction with
     listviews with GTLV_ShowSelected.  (3.1)

   - implemented GTDA_AcceptFunc.  (3.1)

   - if you opened a tree container with open sub-directories the view-list
     order was not correct.  (3.1)

   - GTD_SetAttrs() works now correctly (flags could not be unset).  (3.1)

   - implemented GTDA_TreeView & GTDA_DropOverItems.  (3.1)

   - GTDA_Width & GTDA_Height can now be used for all gadget types.  (3.1)

   - assembler includes added.  (3.1)

   - implemented custom gadget support.  (3.1)

   - add a IFF stream hook.  (3.1)

   - add GTDA_SourceEntry, GTDA_Images now works.  (3.1)

gtdrag 3.2 (9.6.99)
   - add GTDA_InternalOnly for gadgets.  (3.2)

   - if GTDA_TreeView is set, GTDA_Type will be set to ODT_TREENODE, if
     previously unset.  (3.2)

   - unfortunately, ToggleTree() did nothing.  (3.2)

   - the application of a custom gadget is now recognized if its window
     or another gadget in this window is registered.  (3.2)

   - the TNF_STATIC flag of a TreeNode is now supported.  (3.2)

   - documentation enhanced.  (3.2)

   - GTDA_AcceptTypes replaces GTDA_AcceptMask; for your convenience, this
     tag and GTDA_Mask (= GTDA_InternalType) will stay.  (3.2)

   - unfortunately, GadTools copies all incoming messages - a GTD_ReplyIMsg()
```

```
    is therefore an obligate call. This caused a strange bug where Mungwall
    detects a mis-aligned FreeMem(). This is now fixed.  (3.2)

  - the object acceptance cache was not cleared after a drag.  (3.2)

  - the GTDA_DropOverItems flag could cause highlighting problems.  (3.2)

  - flag
              GTDA_DropBetweenItems
               added
    (suggestion made by Guido Mersmann).  (3.2)
```

1.45 gtdrag - Known Bugs

```
Unfortunately, there are bugs left:

 · on CyberGraphX-screens with more than 256 colors the drag object
   is not rendered correctly (true color screens).

 · it is much too slow on a graphic-card; I know the reason for that
   and I know how to fix it, but I do not own a gfx-card and I am too
   busy/lazy to fix it as long I won't have one.

 · the functionality of programs using v2 and older versions is broken
   in v3 - gtdrag simply do nothing in this case. I had no time to
   implement the completely new library behaviour in a more moderate
   way. Since I personally know only one person who use this library
   I can live with it.

 · gadgets with the GFLG_RELBOTTOM & GFLG_RELRIGHT set are not yet
   supported. This may change in future releases.
```

1.46 gtdrag - To do list

```
                There are a few things on the "to do"-list:

 · better CyberGraphX/Picasso96 support (see
              known bugs
              )

 · Drag&Drop between applications on different screens

 · support for gadgets with GFLG_RELBOTTOM or GFLG_RELRIGHT set

That's not all, but I obviously forgot the rest.
```