in

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 baseclass.guide

```
 Search
TABLE OF CONTENTS

baseclass/--background--
baseclass/BASE_ADDCONDITIONAL
baseclass/BASE_ADDHOOK
baseclass/BASE_ADDMAP
baseclass/BASE_ADDMETHOD
baseclass/BASE_CHECKLOOP
baseclass/BASE_CLEARLOOP
baseclass/BASE_DRAGACTIVE
baseclass/BASE_DRAGGING
baseclass/BASE_DRAGINACTIVE
baseclass/BASE_DRAGQUERY
baseclass/BASE_DRAGUPDATE
baseclass/BASE_DROPPED
baseclass/BASE_FINDRAWKEY
baseclass/BASE_FREEDRAGOBJECT
baseclass/BASE_GETDRAGOBJECT
baseclass/BASE_LEFTEXT
baseclass/BASE_RELAYOUT
baseclass/BASE_REMCONDITIONAL
baseclass/BASE_REMHOOK
baseclass/BASE_REMMAP
baseclass/BASE_REMMETHOD
baseclass/BASE_SETLOOP
baseclass/BASE_SHOWHELP
baseclass/BT_BottomOffset
baseclass/BT_Buffer
baseclass/BT_DragObject
baseclass/BT_DragQualifier
baseclass/BT_DragThreshold
baseclass/BT_DropObject
baseclass/BT_HelpFile
baseclass/BT_HelpHook
baseclass/BT_HelpLine
baseclass/BT_HelpNode
baseclass/BT_HelpText
```

baseclass/BT_HelpTextID
baseclass/BT_HitBox
baseclass/BT_Inhibit
baseclass/BT_InnerBox
baseclass/BT_Key
baseclass/BT_LabelClick
baseclass/BT_LabelObject/BT_FrameObject
baseclass/BT_LeftOffset
baseclass/BT_MouseActivation
baseclass/BT_NoRecessed
baseclass/BT_OuterBox
baseclass/BT_PostRenderHighestClass
baseclass/BT_Qualifier
baseclass/BT_RawKey
baseclass/BT_RightOffset
baseclass/BT_TextAttr
baseclass/BT_ToolTip
baseclass/BT_ToolTipID
baseclass/BT_TopOffset
baseclass/GM_GOACTIVE
baseclass/GM_GOINACTIVE
baseclass/GM_HITTEST
baseclass/GM_RENDER
baseclass/OM_NOTIFY
baseclass/OM_UPDATE

1.2 baseclass/--background--

NAME

Class: baseclass
Superclass: GADGETCLASS
Include File: <libraries/bgui.h>

FUNCTION

This is the most important gadget class in BGUI. It is the superclass of almost all the other BGUI gadget classes at one point or another. It will take care of the following things for it's subclasses:

- Notification
- Frames
- Labels
- Online-help.
- Basic Drag and Drop support.

If you plan on writing a gadget class for BGUI then always sub-class it from this class. That way you automatically inherit the above mentioned features.

NOTE

The methods and attributes described in this file are also valid for all other BGUI gadget classes subclassed from this class unless

documented otherwise.

1.3 baseclass/BASE_ADDCONDITIONAL

NAME

BASE_ADDCONDITIONAL -- Add an object for conditional notification.

SYNOPSIS

```
err = DoMethod( obj, CM_ADDCONDITIONAL, target, ctag, ttag, ftag );

ULONG             err;
Object           *target;
struct TagItem    ctag;
struct TagItem    ttag;
struct TagItem    ftag;
```

FUNCTION

To add an object to this object's conditional notification list. Conditional notification is a simple "if ctag = true then set ttag else set ftag" style of notification.

INPUTS

target - A pointer to the object to add.

ctag - A struct TagItem telling the attribute and the value of the attribute data to make the condition TRUE.

ttag - A struct TagItem containing the attribute and attribute data which is set on the target when 'ctag' is TRUE.

ftag - A struct TagItem containing the attribute and attribute data which is set on the target when 'ctag' is FALSE.

RESULT

err - TRUE for success, FALSE for failure.

EXAMPLE

```
Object      *cycle, *string;

/*
 * This will enable the string object when the
 * CYC_Active attribute of the cycle object is
 * 0. When the CYC_Active attribute is not 0
 * the string object is disabled.
 */

DoMethod( cycle, BASE_ADDCONDITIONAL, string,
   CYC_Active,  0,
   GA_Disabled, FALSE,
   GA_Disabled, TRUE );
```

```
SEE ALSO

   BASE_REMCONDITIONAL
```

1.4 baseclass/BASE_ADDHOOK

```
NAME

   BASE_ADDHOOK -- Add an object for hook notification.

SYNOPSIS

   err = DoMethod( obj, BASE_ADDHOOK, hook );

   ULONG          err;
   struct Hook   *hook;

FUNCTION

   To add a hook to this object's hook notification list. Every time a
   notification event occurs the hook is called with the following parameters:

   rc = hookFunc( hook, obj, update );
   D0             A0    A2   A1

   ULONG             rc;
   struct Hook      *hook;
   Object           *obj;
   struct opUpdate  *update;

   hook     -  A pointer to the hook structure.

   obj      -  This is a pointer to the object doing the notification.

   update   -  This is a pointer to an opUpdate structure from which you can
               extract data like the visual environment and the attributes which
               caused the notification.

   rc       -  Your hook routine must return non-zero when the notification
               caused a change of some sort and zero if not.

INPUTS

   hook     -  A pointer to the hook structure to add.

RESULT

   err   -  TRUE for success, FALSE for failure.

EXAMPLE

   /*
    * Stupid display beeping hook.
    */
```

```
   __saveds __asm ULONG beep( register __a0 struct Hook   *hook,
                              register __a2 Object          *obj,
                              register __a1 struct opUpdate *opu ) {
      /*
       * Only final messages.
       */
      if ( ! ( opu->opu_Flags & OPUF_INTERIM )) {
         DisplayBeep(0);
         return( 1 );
      }
      return( 0 );
   }

   struct Hook beep_hook = { NULL,NULL,(HOOKFUNC)beep,NULL,NULL };

   /*
    * Make the button beep when released.
    */
   DoMethod( button, BASE_ADDHOOK, &beep_hook );
```

SEE ALSO

BASE_REMHOOK

1.5 baseclass/BASE_ADDMAP

NAME

BASE_ADDMAP -- Add an object for maplist notification.

SYNOPSIS

```
   err = DoMethod( obj, BASE_ADDMAP, target, maplist );

   ULONG             err;
   Object           *target;
   struct TagItem   *maplist;
```

FUNCTION

To add an object to this object's maplist notification list. Maplist notification is similar to the notification performed by the system "icclass" and ICA_TARGET notification.

INPUTS

target - A pointer to the object to add.

maplist - This can point to an array of TagItem structures containing a set of attributes to map. This may also be NULL in which case no mapping is done.

RESULT

err - TRUE for succes, FALSE for failure.

```
EXAMPLE

   /*
    * This map-list will convert the SLIDER_Level attribute
    * to a INDIC_Level attribute before the attributes are
    * passed to the 'indicator' object.
    */
   struct TagItem map[] = {
      SLIDER_Level, INDIC_Level,
      TAG_END
   };

   DoMethod( slider, BASE_ADDMAP, indicator, map );

SEE ALSO

   BASE_REMMAP
```

1.6 baseclass/BASE_ADDMETHOD

```
NAME

   BASE_ADDMETHOD -- Add an object for method notification.

SYNOPSIS

   err = DoMethod( obj, BASE_ADDMETHOD, target, flags, size, id, ... );

   ULONG    err;
   Object  *target;
   ULONG    flags;
   ULONG    size;
   ULONG    id;

FUNCTION

   To add an object to this object's method notification list.  Method
   notification allows you to send complete methods to another object when a
   notification event occures.

   Please note that most notification will occur on the input.device's task.
   This means that the called method must not take too long to return and
   generally should not do anything a normal input handler or task based code
   should not do.

INPUTS

   target   -  A pointer to the object to add.

   flags    -  Any of the follwoing flags:

               BAMF_NO_GINFO

                  Normally a pointer to a GadgetInfo structure is inserted in
```

```
                the method to send. When this flag is set this will not be
                done. Please note that when a GadgetInfo is placed it is done
                as follows:

                OM_NEW, OM_SET, OM_UPDATE, OM_NOTIFY

                   If the method to send is any of these the GadgetInfo is
                   placed in the third long word of the method. All other
                   methods will get the GadgetInfo in the second long word.
                   This behaviour is the same as with the DoGadgetMethodA()
                   routine from intuition.

             BAMF_NO_INTERIM

                When set the method will not be sent on interim notification
                events.

   size     -  The size in long-words _including_  the method ID of the method
               to send. This information is needed because a copy of the method
               to send is made in an internal buffer.

   id       -  The method ID of the method to send. This field can be followed
               by the method specific data.

RESULT

   err   -  TRUE for succes, FALSE for failure.

EXAMPLE

   /*
    * When notification occures 'obj1' will send the
    * LVM_ADDSINGLE method to 'obj2'. The method will
    * only be sent on final notification and will include
    * a pointer to a GadgetInfo structure.
    */
   DoMethod( obj1, BASE_ADDMETHOD, obj2, BAMF_NO_INTERIM, 5,
               LVM_ADDSINGLE, NULL, "new", LVAP_TAIL, 0 );

SEE ALSO

   BASE_REMMETHOD
```

1.7 baseclass/BASE_CHECKLOOP

```
NAME

   BASE_CHECKLOOP -- Private!
```

1.8 baseclass/BASE_CLEARLOOP

```
NAME

   BASE_CLEARLOOP -- Private!
```

1.9 baseclass/BASE_DRAGACTIVE

```
NAME

   BASE_DRAGACTIVE -- Tell an object to become active.

   ** V40 **

SYNOPSIS

   DoMethod( obj, BASE_DRAGACTIVE, gi, source );

   struct GadgetInfo *gi;
   Object            *source;

FUNCTION

   When the object accepted to become the drag object's target this method is
   called. It tells the object that it has become the active drop object by
   sending this method.

   When this method is called the object should make some visual changes to
   itself to let the user know that it will accept a drop from the dragged
   object. By default the baseclass will render a dotted box around the hit-box
   of the object.

   When you are not satisfied with this look or you simply need another visual
   confirmation your class should overide the baseclass and do it itself.

INPUTS

   gi       -  A pointer to the GadgetInfo structure.

   source   -  A pointer to the object making this abject active.

RESULT

   The object should make it visually obvious it has become the active drop
   object. A return code is not defined.

SEE ALSO

   BASE_DRAGQUERY, BASE_DRAGUPDATE, BASE_DRAGINACTIVE,

   The DragNDrop demos.
```

1.10 baseclass/BASE_DRAGGING

NAME

BASE_DRAGGING -- Input handling method for DragNDrop.

** V40 **

SYNOPSIS

```
rc = DoMethodA( obj, gpi );

ULONG             rc;
struct gpInput   *gpi;
```

FUNCTION

To enable the baseclass to inform its subclasses when it is dragging an object around. I had to implement a special method which basically is the same as the normal GM_HANDLEINPUT method supplied by the system.

Because I had to have the option to tell the subclasses what to do I could not use the GM_HANDLEINPUT method for compatibility reasons. This is where this method comes in.

When you write a class which uses Drag and Drop as an option the first thing you must do in your GM_HANDLEINPUT routine is call this method. Depending on what this method returns you continue processing the input or not.

This method uses the same structure as the GM_HANDLEINPUT method does (struct gpInput *).

INPUTS

gpi - A pointer to a gpInput structure. Please note that the MethodID field must be set to BASE_DRAGGING (see example).

RESULT

rc - Any of the following return codes:

BDR_NONE

When this is returned the baseclass did not do anything with the message and you should process it yourself like you would normally.

BDR_DRAGPREPARE

This result tells you to prepare for the fact that the baseclass is going to take over the input handling. For example a draggable button might deselect itself before the dragging starts.

BDR_DRAGGING

When this is returned it means that the baseclass is moving the object around. You should do nothing except for returning GMR_MEACTIVE. Do not handle the input event.

BDR_DROP

```
            This return code means that the object has been dropped on a
            droppable object. Usually this means that you stop input processing
            and return GMR_NOREUSE.

         BDR_CANCEL

            This return code means that either the drag has been cancelled or
            the user dropped the object somewhere outside a droppable object.
            Usually this means you return GMR_NOREUSE.

EXAMPLE

   /*
    * A simple example of a drag prepared
    * GM_HANDLEINPUT routine could be this:
    */
   ULONG handle_input( cl, obj, gpi )
      Class        *cl;
      Object          *obj;
      struct gpInput *gpi;
   {
      struct gpInput copy  = *gpi;
      ULONG          rc    =  GMR_MEACTIVE;

      copy.MethodID = BASE_DRAGGING;
      switch ( DoMethodA( obj, ( Msg )&copy )) {

         case  BDR_NONE:
            /* Process input like ususal. */
            break;

         case  BDR_DROP:
         case  BDR_CANCEL:
            rc = GMR_NOREUSE;
            break;

         case  BDR_DRAGPREPARE:
            /* Unselect button or whatever you
               need to do before dragging
               starts. */
            break;
      }
      return( rc );
   }

SEE ALSO

   The DragNDrop demos.
```

1.11 baseclass/BASE_DRAGINACTIVE

```
NAME

   BASE_DRAGINACTIVE -- Deactivate drop target.
```

```
   ** V40 **

SYNOPSIS

   DoMethod( obj, BASE_DRAGINACTIVE, gi, source );

   struct GadgetInfo *gi;
   Object            *source;

FUNCTION

   When the object is no longer the active target of the dragged object this
   method is called.

   In this method the visual changes made in the BASE_DRAGACTIVE method must be
   undone.

   By default the baseclass simply re-renders the object. If you are not content
   with this behaviour your class should override the baseclass and do it
   itself.

INPUTS
   gi       - A pointer to the GadgetInfo structure.

   source   - A pointer to the object deactivating you.

RESULT

   The visual changes made in BASE_DRAGACTIVE should be undone. No return code
   has been defined for this method.

SEE ALSO

   BASE_DRAGQUERY, BASE_DRAGACTIVE, BASE_DRAGUPDATE,

   The DragNDrop demos.
```

1.12 baseclass/BASE_DRAGQUERY

```
NAME

   BASE_DRAGQUERY -- Query if an object is droppable.

   ** V40 **

SYNOPSIS

   rc = DoMethod( obj, BASE_DRAGQUERY, gi, source, xy );

   ULONG                rc;
   struct   GadgetInfo *gi;
   Object              *source;
   LONG                 xy;
```

FUNCTION

This method is sent to the dropable object when another object is dragged over it. It is used to find out whether or not the object is prepared or willing to become the active target of the dragged object.

INPUTS

gi - A pointer to the drop-object it's GadgetInfo.

source - A pointer to the object wanting to know if this object is willing to become the active drop target.

xy - A 32 bit integer containing the mouse coordinates relative to the top-left of the _HitBox_ from this object. With this information you can opt to become active only when the mouse is at a certan position. The ListviewClass uses this to only become the active drop target if the mouse is in the list area. The upper 16 bits contain the x offset and the lower 16 bits the y offset.

RESULT

rc - When this object is ready and willing to accept data from the inquiring object it should return BQR_ACCEPT. If not it should return BQR_REJECT.

NOTE

The baseclass defaults to BQR_ACCEPT for _all_ drag objects which are dragged uppon its HitBox. This means that to implement a usable droppable object you must write a subclass which overides this method to make a more resonable selection.

SEE ALSO

BASE_DRAGACTIVE, BASE_DRAGINACTIVE, The DragNDrop demos.

1.13 baseclass/BASE_DRAGUPDATE

NAME

BASE_DRAGUPDATE -- Update drag position.

** V40 **

SYNOPSIS

```
rc = DoMethod( obj, BASE_DRAGUPDATE, gi, source, xy );

ULONG                rc;
struct   GadgetInfo *gi;
Object              *source;
LONG                 xy;
```

FUNCTION

When the object has become the active drop object it will get continues updates from the dragged object about the mouse pointer position relative to the top-left corner of it's hitbox. The updates are sent on each timer tick.

The update messages can be very usefull if your class needs to know the place where the object is dropped. For instance the ListviewClass uses these message to update the visuals to make clear where the dragged entry or entries will be dropped.

INPUTS

gi - A pointer to the GadgetInfo structure.

source - A pointer to the object which sends the updates.

xy - A 32 bit integer containing the mouse position relative to the top-left corner of the hitbox. The upper 16 bits contain the x offset and the lower 16 bits the y offset.

RESULT

rc - One of the following return codes:

BUR_CONTINUE

To signal the dragged object that this object wants to remain the active target.

BUR_ABORT

To signal the dragged object that this object no longer wants to be the active target.

NOTE

The base class will return BUR_CONTINUE as long as the mouse is still located inside the object it's hitbox. When the mouse has been moved outside the object hitbox the baseclass will return BUR_ABORT.

SEE ALSO

BASE_DRAGACTIVE, BASE_DRAGINACTIVE, The DragNDrop demos.

1.14 baseclass/BASE_DROPPED

NAME

BASE_DROPPED -- The object has been dropped.

** V40 **

SYNOPSIS

DoMethod(obj, BASE_DROPPED, gi, source, win, req);

```
   struct   GadgetInfo *gi;
   Object              *source;
   struct   Window     *win;
   struct   Requester  *req;
```

FUNCTION

This method is called when the user has dropped the dragged object on this object. Now this object may get the data it needs from the object and act accordingly.

INPUTS

gi - A pointer to the GadgetInfo.

source - A pointer to the object which has been dropped.

win - A pointer to the window on which the dropped object is located. With this pointer you can use BGUI_DoGadgetMethod() to make visual changes to the dropped object if necessary.

req - A pointer to the requester on which the dropped object is located. Currently BGUI does not support requesters but for future compatibility you should pass this pointer along with any BGUI_DoGadgetMethod() you make.

RESULT

The object should do whatever needs to be done with the dropped object. There is no return code defined for this method.

NOTE

To enable a useful drag and drop object you should write a subclass which will overide the baseclass and act on dropped objects. The baseclass simply beep the display.

SEE ALSO

bgui.library/BGUI_DoGadgetMethod(), The DragNDrop demos.

1.15 baseclass/BASE_FINDRAWKEY

NAME

BASE_FINDRAWKEY -- Match a rawkey to an object (or subobjects).

** V41 **

SYNOPSIS

```
ob = DoMethod( obj, BASE_FINDRAWKEY, rawkey );

Object  *ob;
```

```
   UBYTE    rawkey;

FUNCTION

   To see if an object has the specified rawkey as its key equivalent. If the
   object is a grouping or paging object, all members will checked against the
   specified rawkey. A pointer to an object that matches the rawkey is returned,
   or NULL if there is no match.

   This method is called by the windowclass during input handling, and is mainly
   useful to class writers. If you implement a class that contains multiple
   objects that can have independent key equivalents, you will need to define
   this method for your class, and iterate through each sub-object in your
   object, calling BASE_RAWKEY for each one. If you get a non-NULL return value,
   return it. If no match is found, return NULL.

INPUTS

   rawkey   -  The rawkey to match.

RESULT

   ob -  Object with the specified rawkey, or NULL if none.

EXAMPLE

   UBYTE    k;
   Object  *group;

   /*
    * This would show what object is attached to each function key in the group.
    */
   for (k = 0x50; k <= 0x59; k++)
   {
      printf("F%ld: %08lx\n", k - 0x50 + 1, DoMethod(group, BASE_FINDRAWKEY, k));
   };

SEE ALSO
```

1.16 baseclass/BASE_FREEDRAGOBJECT

```
NAME

   BASE_FREEDRAGOBJECT -- Delete the bitmap draggged around.

   ** V40 **

SYNOPSIS

   DoMethod( obj, BASE_FREEDRAGOBJECT, bm );

   struct BitMap *bm;

FUNCTION
```

This method is called to free the BitMap created with the BASE_GETDRAGOBJECT method.

INPUTS

bm - A pointer to the BitMap structure as returned by the BASE_GETDRAGOBJECT method.

RESULT

The bitmap must be deallocated. No return code defined.

NOTE

When your class overiddes the BASE_GETDRAGOBJECT method it should also override this method.

SEE ALSO

BASE_GETDRAGOBJECT

1.17 baseclass/BASE_GETDRAGOBJECT

NAME

BASE_GETDRAGOBJECT -- Obtain bitmap to drag.

** V40 **

SYNOPSIS

```
bm = DoMethod( obj, BASE_GETDRAGOBJECT, gi, ibox );

struct   BitMap     *bm;
struct   GadgetInfo *gi;
struct   IBox       *ibox;
```

FUNCTION

This method is sent to the object when this class needs to obtain a bitmap containing the image to drag accross the screen.

INPUTS

gi - A pointer to the GadgetInfo structure.

ibox - A pointer to an IBox structure in which you should store the bounds of the bitmap you create. The Left and Top fields are used by the baseclass to buffer the screen information which is going to be covered by the dragged object. The Width and Height should be the width and height of the bitmap.

RESULT

bm - A pointer to the BitMap structure containing the image to drag around

```
         or NULL upon failure.

NOTE

   By default the baseclass will return a pointer to a BitMap containing the
   Hitbox imagery of the object. If you want to change this your class should
   override the baseclass and supply its own bitmap. The PaletteClass example
   does this.

SEE ALSO

   BASE_FREEDRAGOBJECT, PaletteClass demos.
```

1.18 baseclass/BASE_LEFTEXT

```
NAME

   BASE_LEFTEXT -- Private!
```

1.19 baseclass/BASE_RELAYOUT

```
    NAME
  BASE_RELAYOUT -- Force an object to layout again.

    SYNOPSIS
  BGUI_DoGadgetMethod( obj, bmRelayout);

  Object    *obj;
  struct bmRelayout *bmRelayout;

    FUNCTION
  To force a object in an opened window to layout again.  If the object
  does not fit in the current group it may force all the parent groups to
  resize and eventually the window on which it is displayed.

    INPUTS
  obj - A pointer to the object which is to be layed out again.
  bmRelayout - A pointer to a BASE_RELAYOUT method structure.

    RESULT
  Returns TRUE if the window did not need to resize or if the after
  resizing it still fits on the screen.

    SEE ALSO
```

1.20 baseclass/BASE_REMCONDITIONAL

```
NAME

   BASE_REMCONDITIONAL -- Remove an conditional notification object.
```

```
SYNOPSIS

   err = DoMethod( obj, BASE_REMCONDITIONAL, target );

   ULONG    err;
   Object  *target;

FUNCTION

   To remove an object from the conditional notification list which was
   previously added with BASE_ADDCONDITIONAL.

INPUTS

   target   -  A pointer to the object to remove which was previously added with
               the BASE_ADDCONDITIONAL method.

RESULT

   err   -  TRUE for success, FALSE for failure.

EXAMPLE

   /*
    * Remove 'obj2' from 'obj1' its conditional
    * notification list.
    */
   Domethod( obj1, BASE_REMCONDITIONAL, obj2 );

SEE ALSO

   BASE_ADDCONDITIONAL
```

1.21 baseclass/BASE_REMHOOK

```
NAME

   BASE_REMHOOK -- Remove a notification hook.

SYNOPSIS

   err = DoMethod( obj, BASE_REMHOOK, hook );

   ULONG          err;
   struct   Hook *hook;

FUNCTION

   To  remove a hook from the hook notification list which was previously added
   with BASE_ADDHOOK.

INPUTS

   hook  -  A pointer to the hook to remove which was previously added with the
```

```
            BASE_ADDHOOK method.

RESULT

   err   -  TRUE for success, FALSE for failure.

EXAMPLE
   /*
    * Remove 'hook' from 'obj' its hook
    * notification list.
    */
   Domethod( obj, BASE_REMHOOK, hook );

SEE ALSO

   BASE_ADDHOOK
```

1.22 baseclass/BASE_REMMAP

```
NAME

   BASE_REMMAP -- Remove a maplist notification object.

SYNOPSIS

   err = DoMethod( obj, BASE_REMMAP, target );

   ULONG    err;
   Object  *target;

FUNCTION

   To remove an object from the maplist notification list which was previously
   added with BASE_ADDMAP.

INPUTS

   target   -  A pointer to the object to remove which was previously added with
               the BASE_ADDMAP method.

RESULT

   err   -  TRUE for success, FALSE for failure.

EXAMPLE
   /*
    * Remove 'obj2' from 'obj1' its maplist
    * notification list.
    */
   Domethod( obj1, BASE_REMMAP, obj2 );

SEE ALSO

   BASE_ADDMAP
```

1.23 baseclass/BASE_REMMETHOD

```
NAME

   BASE_REMMETHOD -- Remove a method notification object.

SYNOPSIS

   err = DoMethod( obj, BASE_REMMETHOD, target );

   ULONG    err;
   Object  *target;

FUNCTION

   To remove an object from the method notification list which was previously
   added with BASE_ADDMETHOD.

INPUTS

   target   -  A pointer to the object to remove which was previously
               added with the BASE_ADDMETHOD method.

RESULT

   err   - TRUE for success, FALSE for failure.

EXAMPLE

   /*
    * Remove 'obj2' from 'obj1' its method
    * notification list.
    */
   Domethod( obj1, BASE_REMMETHOD, obj2 );

SEE ALSO

   BASE_ADDMETHOD
```

1.24 baseclass/BASE_SETLOOP

```
NAME

   BASE_SETLOOP -- Private!
```

1.25 baseclass/BASE_SHOWHELP

```
NAME

   BASE_SHOWHELP -- Private!
```

1.26 baseclass/BT_BottomOffset

```
NAME

  BT_BottomOffset -- ( ULONG )

FUNCTION

  This is a gap just inside the enclosing frame for the gadget,
  measured in pixels.

  The default value of 0 means that the outer frame encloses the
  gadget "tightly", with no gap in between.

  Values higher than zero will give you a gap.

  eg. setting this to 10 will mean there is a 10 pixel high gap
  between the bottom of the frame and the InnerBox of your
  gadget.

DEFAULT

  0.  (no gap)

APPLICABILITY

  (ISG).

SEE ALSO

  BT_LeftOffset, BT_RightOffset, BT_TopOffset
```

1.27 baseclass/BT_Buffer

```
NAME

   BT_Buffer -- ( ULONG )

   ** V41.6 **

FUNCTION

   To be documented.

DEFAULT

   To be documented.

APPLICABILITY

   (ISG).

SEE ALSO
```

1.28 baseclass/BT_DragObject

```
NAME

   BT_DragObject -- ( BOOL )

   ** V40 **

FUNCTION

   To enable the object to be dragged around the screen and dropped on other
   objects by the user.

DEFAULT

   FALSE.

APPLICABILITY

   (ISG).

SEE ALSO

   BT_DropObject, BT_DragThreshold, BT_DragQualifier
```

1.29 baseclass/BT_DragQualifier

```
NAME

   BT_DragQualifier -- ( UWORD )

   ** V40 **

FUNCTION

   To specify the qualifier of the input event that must be present to allow
   dragging. For example you might choose to support the dragging of an object
   only if the left alt key is pressed while selecting the object. A simple

      BT_DragQualifier, IEQUALIFIER_LALT

   will do this.

   Have a look at the "devices/inputevent.[h][i]" file for the available
   possibilities.

DEFAULT

   ~0 (no qualifier).
```

APPLICABILITY

(ISG).

SEE ALSO

BT_DragObject, BT_DropObject, BT_DragThreshold, devices/inputevent.h

1.30 baseclass/BT_DragThreshold

NAME

BT_DragThreshold -- (ULONG)

** V40 **

FUNCTION

To set the number of pixels that the mouse must be moved before the object is actually dragged.

DEFAULT

3.

APPLICABILITY

(ISG).

SEE ALSO

BT_DragObject, BT_DropObject, BT_DragQualifier

1.31 baseclass/BT_DropObject

NAME

BT_DropObject -- (BOOL)

** V40 **

FUNCTION

To enable the object to receive messages from a draggable object.

DEFAULT

FALSE.

APPLICABILITY

(ISG).

SEE ALSO

BT_DragObject, BT_DragThreshold, BT_DragQualifier

1.32 baseclass/BT_HelpFile

NAME

BT_HelpFile -- (STRPTR)

FUNCTION

Set the name of the file to be displayed when a help-request for the object arives. Please note that the full path-name must be given.

DEFAULT

NULL.

APPLICABILITY

(ISG).

SEE ALSO

BT_HelpNode, BT_HelpLine

1.33 baseclass/BT_HelpHook

NAME

BT_HelpHook -- (ULONG)

** V 41.7 **

FUNCTION

To be documented.

DEFAULT

To be documented.

APPLICABILITY

(ISG).

SEE ALSO

1.34 baseclass/BT_HelpLine

NAME

BT_HelpLine - (ULONG)

FUNCTION

Set the line number from which the file is displayed. This may be useful if the help-file is not an AmigaGuide file.

DEFAULT

0.

APPLICABILITY

(ISG).

SEE ALSO

BT_HelpFile, BT_HelpNode

1.35 baseclass/BT_HelpNode

NAME

BT_HelpNode -- (STRPTR)

FUNCTION

Set the name of the node which is diplayed in the help window.

DEFAULT

NULL.

APPLICABILITY

(ISG).

SEE ALSO

BT_HelpFile, BT_HelpLine

1.36 baseclass/BT_HelpText

NAME

BT_HelpText -- (STRPTR)

FUNCTION

```
   To setup a text which will be displayed if the help-key is pressed while the
   mouse pointer is located above the object. This attribute should be used to
   attach small on-line help to the object. The text you specify will be shown
   in a small BGUI_RequestA() type of requester so you must make sure that
   everything fit's nicely on a 600x200 screen.

   The specified text may contain any of the infoclass command sequences.

   This attribute overides the BT_HelpFile, BT_HelpNode and BT_HelpLine
   attributes. Also note that the requester which pops up is synchronous.

DEFAULT

   NULL.

APPLICABILITY

   (ISG).

SEE ALSO

   BT_HelpFile, BT_HelpNode, BT_HelpLine, infoclass.doc/INFO_TextFormat,
   bgui.library/BGUI_RequestA()
```

1.37 baseclass/BT_HelpTextID

```
NAME

   BT_HelpTextID -- ( ULONG )

   ** V41 **

FUNCTION

   Set or get the ID for BT_HelpText. BASE_LOCALIZE uses this to set
   BT_HelpText.

DEFAULT

   0.

APPLICABILITY

   (ISG).

SEE ALSO

   BT_HelpText, baseclass.doc/BASE_LOCALIZE
```

1.38 baseclass/BT_HitBox

NAME

BT_HitBox - (struct IBox *)

FUNCTION

Get the hitbox bounds of the gadget object. This attribute is normally only useful for class writers although there might be cases in which this attribute may be useful to application programmers to.

Getting this attribute will pass you a pointer to a _READ ONLY_ struct IBox in which the bounds of the object are located.

Please note that the contents of the returned IBox structure is only valid _after_ you let this class render itself.

DEFAULT

None.

APPLICABILITY

(G).

EXAMPLE

```
__saveds __asm ULONG
dispatcher ( __a0 Class *cl, __A2 Object *obj, __A1 Msg msg )
{
   struct IBox    *domain;

   switch ( msg->MethodID ) {

      ...

      case  GM_RENDER:
         /*
         ** First let the superclass render.
         **/
         if ( ! DoSuperMethodA( cl, obj, msg ))
            return( 0L );
         /*
         ** Now you can obtain the object
         ** it's hitbox bounds.
         **/
         DoSuperMethod( cl, obj, OM_GET,
                   BT_HitBox, &domain );

         ...

         break;
   }
}
```

1.39 baseclass/BT_Inhibit

NAME

BT_Inhibit -- (BOOL)

** V39 **

FUNCTION

This attribute has been made public for people needing to know whether or not an object is located on a visible page. This attribute is read only and will read TRUE when an object is located on an invisible page and FALSE if the object is located on a visible page.

DEFAULT

FALSE.

APPLICABILITY

(G).

1.40 baseclass/BT_InnerBox

NAME

BT_InnerBox -- (ULONG)

FUNCTION

To be documented.

DEFAULT

To be documented.

APPLICABILITY

(G).

SEE ALSO

BT_OuterBox

1.41 baseclass/BT_Key

NAME

BT_Key -- (STRPTR)

** V41 **

```
FUNCTION

   To attach a key equivalent to an object. As of V41.3, events are processed as
   IDCMP_VANILLAKEY, so any key may be attached, including characters that
   require qualifiers and deadkeys.

   It is usually not desirable to specify a qualifier with BT_Qualifier, but it
   is possible. This could be useful to distinguish between a a number pressed
   on the main keyboard and one on the numeric keypad, for example.

DEFAULT

   NULL.

APPLICABILITY

   (ISG).

SEE ALSO

   BT_Qualifier, BT_RawKey, BASE_FINDRAWKEY, BASE_KEYLABEL,
   windowclass.doc/WINDOW_AutoKeyLabel
```

1.42 baseclass/BT_LabelClick

```
NAME

   BT_LabelClick -- ( BOOL )

FUNCTION

   To tell the baseclass to also consider clicking inside the gadget
   label as a hit.

DEFAULT

   FALSE.

APPLICABILITY

   (I).

SEE ALSO

   GM_HITTEST
```

1.43 baseclass/BT_LabelObject/BT_FrameObject

```
NAME

   BT_LabelObject, BT_FrameObject -- ( Object * )
```

```
FUNCTION

   Set or get the frame/label object to use. Normally only class writers use
   these attributes. You can use them to obtain a pointer to the label/frame
   object or to erase/change the label/frame object. If, for example, your class
   uses its own custom rendering you can dispose of the baseclass frame by
   setting it to NULL like this:

      SetAttrs( object, FRM_FrameObject, NULL, TAG_END );

DEFAULT

   The frame/label build from the attributes passed to the object at create
   time.

APPLICABILITY

   (SG).

SEE ALSO

   intuition.library/SetAttrs()
```

1.44 baseclass/BT_LeftOffset

```
NAME

   BT_LeftOffset -- ( ULONG )

   ** V ? **

FUNCTION

  This is a gap just inside the enclosing frame for the gadget,
  measured in pixels.

  The default value of 0 means that the outer frame encloses the
  gadget "tightly", with no gap in between.

  Values higher than zero will give you a gap.

  eg. setting this to 10 will mean there is a 10 pixel wide gap
  between the left of the frame and the InnerBox of your
  gadget.

DEFAULT

  0.  (no gap)

APPLICABILITY

   (ISG).

SEE ALSO
```

BT_RightOffset, BT_TopOffset, BT_BottomOffset

1.45 baseclass/BT_MouseActivation

NAME

BT_MouseActivation -- (ULONG)

** V41.5 **

FUNCTION

This flag/attribute allows the gadget to respond to right and middle mouse buttons. By setting BT_MouseActivation to some combination of the following flags, you can allow the gadget to hear mouse button clicks from any of the buttons, left, middle, right.

The following values may be ORed together:

· MOUSEACT_RMB_ACTIVE
The right mouse button will activate the gadget just like the left mouse button.

· MOUSEACT_RMB_REPORT
This is to make the right mouse button cause the gadget ID to be reported in a notification event.

· MOUSEACT_MMB_ACTIVE same as above except for the middle mouse button.
· MOUSEACT_MMB_REPORT

BOOPSI gadgets are not supposed to handle right mouse events until the gadget becomes active. However, if the window class gets right mouse button events when the mouse is over a gadget with this attribute set, the gadget is activated.

This way we can implement a context sensitive menu or another user interface action being triggered by the the right or middle mouse button.

It is actually the Window class that makes use of this attribute, allowing whatever mouse button events you are interested in through to your gadget. BGUI Window class handles Intuition events for us.

To let this happen, the Window must be getting mouse events.

How do you specify that a Window should get mouse events?

You have to set IDCMP_MOUSEMOVE and IDCMP_MOUSEBUTTONS for the Window IDCMP flags but BGUI sets that for you.

When does a Window not get mouse events?

- When the window is not active
- When the user pulls down an Intuition menu and the Window does not have RMBTrap set.

NOTE

The left mouse button is most usually the "Select" button, and the right mouse button the "Menu" button. This can be swapped, without physically changing the mouse! The input events the gadget receives are SELECTDOWN, SELECTUP, MENUDOWN, and MENUUP.

DEFAULT

0 (zero). (only activate on left mouse presses/releases)

APPLICABILITY

(ISG)

SEE ALSO

bgui.h for the definition of this attribute and the flags above.

1.46 baseclass/BT_NoRecessed

NAME

BT_NoRecessed -- (BOOL)

FUNCTION

To tell the baseclass not to recess the frame when the gadget object is selected. The checkboxclass uses this attribute.

DEFAULT

FALSE.

APPLICABILITY

(S).

1.47 baseclass/BT_OuterBox

NAME

BT_OuterBox -- (ULONG)

** V 41.8 **

FUNCTION

To be documented.

DEFAULT

```
   To be documented.

APPLICABILITY

   (G).

SEE ALSO

  BT_InnerBox
```

1.48 baseclass/BT_PostRenderHighestClass

```
NAME

   BT_PostRenderHighestClass -- ( ULONG )

FUNCTION

   To be documented.

DEFAULT

   To be documented.

APPLICABILITY

   (G).

SEE ALSO
```

1.49 baseclass/BT_Qualifier

```
NAME

   BT_Qualifier -- ( UWORD )

   ** V41 **

FUNCTION

   Set the key activation qualifier for the object. When the user presses a key,
   the qualifier must match exactly for the object to be activated. Only
   keyboard qualifiers are considered.

   If you want the object to respond to either shift key, then set both
   IEQUALIFIER_LSHIFT and IEQUALIFIER_RSHIFT, and the object will respond to
   either one or both. You may do this with LALT and RALT as well.

   A value of ~0 will cause qualifiers to be ignored.

DEFAULT
```

```
   ~0.

APPLICABILITY

   (ISG).

SEE ALSO

   BT_Key, BT_RawKey, BASE_FINDRAWKEY, BASE_KEYLABEL,
   windowclass.doc/WINDOW_AutoKeyLabel
```

1.50 baseclass/BT_RawKey

```
NAME

   BT_RawKey -- ( UBYTE )

   ** V41 **

FUNCTION

   To attach a key equivalent to an object. This must be a key-down event.
   IDCMP_VANILLAKEY events take precedence, so use BT_Key unless you need a
   function key, the HELP key, or the cursor keys. You may specify a qualifier
   with BT_Qualifier.

   Note: 0 is a valid rawkey. To remove the rawkey equivalent, set BT_RawKey to
   (UBYTE)~0.

DEFAULT

   ~0.

APPLICABILITY

   (ISG).

SEE ALSO

   BT_Key, BT_Qualifier, BASE_FINDRAWKEY, BASE_KEYLABEL,
   windowclass.doc/WINDOW_AutoKeyLabel
```

1.51 baseclass/BT_RightOffset

```
NAME

   BT_RightOffset -- ( ULONG )

   ** V ? **

FUNCTION
```

```
  This is a gap just inside the enclosing frame for the gadget,
  measured in pixels.

  The default value of 0 means that the outer frame encloses the
  gadget "tightly", with no gap in between.

  Values higher than zero will give you a gap.

  eg. setting this to 10 will mean there is a 10 pixel wide gap
  between the right of the frame and the InnerBox of your
  gadget.

DEFAULT

  0.  (no gap)

APPLICABILITY

   (ISG).

SEE ALSO

  BT_LeftOffset, BT_TopOffset, BT_BottomOffset
```

1.52 baseclass/BT_TextAttr

```
NAME

   BT_TextAttr -- ( struct TextAttr * )

FUNCTION

   Set/Get the font which will be used by the frame and label of the class.
   Class writers might want to intercept and copy this data before passing it
   onto the superclass.

NOTE ** V39 **

   Since V39 this attribute has been made settable at create time. When you
   create an object with BT_TextAttr set the font you pass will be used
   throughout the life of the object. Otherwise the font will be determined by
   the one you set in the windowclass.

DEFAULT

   NULL (Screen font).

APPLICABILITY

   (ISG).
```

1.53 baseclass/BT_ToolTip

```
NAME

   BT_ToolTip -- ( UBYTE * )

   ** V40 **

FUNCTION

   To set the tool tip text for this object. The tool tip is displayed when the
   user holds the mouse still over the object for a specific amount of
   INTUITICKS. The INTUITICKS occur about ten times a second.

   ToolTips should be used on objects for which it is  not immediately obvious
   what it does. Like for instance the objects in graphical tool bars etc.

DEFAULT

   NULL.

APPLICABILITY

   (ISG).
```

1.54 baseclass/BT_ToolTipID

```
NAME

   BT_ToolTipID -- ( ULONG )

   ** V41 **

FUNCTION

   Set or get the ID for BT_ToolTip. BASE_LOCALIZE uses this to set BT_ToolTip.

DEFAULT

   0.

APPLICABILITY

   (ISG).

SEE ALSO

   BT_ToolTip, baseclass.doc/BASE_LOCALIZE
```

1.55 baseclass/BT_TopOffset

```
NAME
```

```
   BT_TopOffset -- ( ULONG )

   ** V ? **

FUNCTION

  This is a gap just inside the enclosing frame for the gadget,
  measured in pixels.

  The default value of 0 means that the outer frame encloses the
  gadget "tightly", with no gap in between.

  Values higher than zero will give you a gap.

  eg. setting this to 10 will mean there is a 10 pixel high gap
  between the top of the frame and the InnerBox of your
  gadget.

DEFAULT

  0.  (no gap)

APPLICABILITY

   (ISG).

SEE ALSO

  BT_LeftOffset, BT_RightOffset, BT_BottomOffset
```

1.56 baseclass/GM_GOACTIVE

```
NAME

   GM_GOACTIVE -- Tell the object to go active.

FUNCTION

   This method will set up any required drag and drop buffers.  When this method
   returns GMR_NOREUSE it means that either drag operations are not supported
   for this object or the available memory will not allow it.

   When GMR_MEACTIVE is returned the drag and drop buffers are set up OK.
```

1.57 baseclass/GM_GOINACTIVE

```
NAME

   GM_GOINACTIVE -- Tell an object to deactivate.

FUNCTION
```

This method will clean up any drag and drop buffers which may have been allocated.

1.58 baseclass/GM_HITTEST

NAME

GM_HITTEST -- Check if the object was "hit".

FUNCTION

This method will return GMR_GADGETHIT when the object was clicked inside it's hitbox area. It will also return GMR_GADGETHIT when the object was clicked inside it label and the BT_LabelClick attribute of this object is set to TRUE.

SEE ALSO

BT_LabelClick

1.59 baseclass/GM_RENDER

NAME

GM_RENDER -- Render the object.

FUNCTION

If this message requests a complete re-rendering (I.E. GREDRAW_REDRAW) this method will render the frame and label when available. On any other request it will simply re-compute the gadget hitbox bounds. When this method returns a non-NULL value you are also allowed to render. If this method returns NULL then you may not render.

Your class _must_ also follow the same rules. If the superclass of your class returns NULL you do not render and also return NULL. If the superclass of your class returns a non-NULL value you should render and also return a non-NULL value.

Your class must use the gpr_RPort field for _all_ rendering. This usually points to a buffer rastport in which you render without it showing on screen. This also means that if your class uses OM_SET on another object you must do so without passing the gpr_GInfo field and re-render that object later in the RastPort pointed to by the gpr_RPort field.

RastPort-Clipping is not allowed for BGUI objects. The reason for this limitation is that all rendering occurs in a buffer rastport without a layer attached to it. If your class absolutely needs clipping you must use the WINDOW_NoBufferRP or PAGE_NoBufferRP on the window object or page object in which the object is located.

1.60 baseclass/OM_NOTIFY

NAME

OM_NOTIFY -- Notify attribute changes.

FUNCTION

This method will first execute all maplist, conditional, method and hook notification. When that is done the method is passed onto GADGETCLASS for any ICA_TARGET's that might still exist.

1.61 baseclass/OM_UPDATE

NAME

OM_UPDATE -- Update attributes.

FUNCTION

This method fixes a bug in the system GADGETCLASS. Even though the BOOPSI gadgetclass documentation states that the GA_ID attribute of a gadget object cannot be changed by a OM_UPDATE message it still does. As this behaviour is not correct the baseclass intercepts the GA_ID attribute before passing it on to the GADGETCLASS. This way GA_ID will not be changable by a OM_UPDATE call.