Appendix: Boopsi Class Reference

This appendix documents all the standard classes, their methods, and their attributes. The parameters for each moethod can be found in the appropriate include file. Each attribute is followed by a code that lists the methods that apply to that attribute. The codes are as follows:

I	`OM_NEW`
S	`OM_SET`
G	`OM_GET`
N	`OM_NOTIFY`
U	`OM_UPDATE`

Class: ***rootclass***
Superclass: None
Description: Base class for all other classes
Include File: *<intuition/classusr.h>*

New Methods:

`OM_NEW` - Create a new object. This method is passed a pointer to the "true class" of the object in place of the yet to be created object. The Root class will allocate enough memory for all the instance data of the true class.

`OM_DISPOSE` - Delete an object. This method will free the same amount of memory as allocated by `OM_NEW`.

`OM_ADDTAIL` - Add an object to an Exec list, based on a MinNode in the private *rootclass* object header. You can provide any Exec list you want, and iterate the objects on the list using the Intuition function NextObject().

`OM_REMOVE` - Remove an object from an Exec list.

The following methods are defined for all objects, but are no-ops in the *rootclass*. Implementation for specific subclasses is done to support these conventions.

`OM_ADDMEMBER` - Add some object (passed to the method) to the objects list, which is defined by the class. Examples include the broadcast list of *modelclass* objects, and the component gadgets of a *groupgclass* object. This is typically implemented by sending the `OM_ADDTAIL` message to the member object.

`OM_REMMEMBER` - Remove a member object previously added by `OM_ADDMEMBER`. You typically send the `OM_REMOVE` message to the member object.

`OM_SET` - Set attributes from passed attribute list. The Root class defines no attributes.

`OM_GET` - Retrieve an attribute.

`OM_UPDATE` - "Be updated" of external changes. A no-op for *rootclass*.

`OM_NOTIFY` - Notify other of changes provided in the message. A no-op for the root class.

Changed Methods:

Not applicable.

Attributes:

None.

Class: ***imageclass***
Superclass: *rootclass*
Description: Base class for Intuition Images
Include File: *<intuition/imageclass.h>*

New Methods:

`IM_DRAW` - Draw the image in the RastPort passed in the message, at the xoffset and in the state provided in the message.

`IM_HITTEST` - Return `TRUE` if the point passed is within the image. This base Image class returns `TRUE` if the point is within the old Image structure box.

`IM_ERASE` - Erase an image. This class will call the new graphics library function EraseRect() for its Image box. *`IM_ERASE` does not test the box dimensions (>= 0).*

`IM_MOVE` - Erase and redraw an image. This is intended for subclasses capable of performing animation or smooth dragging (e.g., prop gadget knobs). A no-op for *imageclass*.

`IM_FRAMEBOX` - A no-op for this base class, but implemented by image subclasses which are "stretchable". For example, *frameiclass* objects put an embossed frame around a text label or glyph. The message passed to this method contains a pointer to a box describing the dimensions of the contents to frame, and a pointer to a frame box for the result. There is also a FrameFlags field which currentlly has only one defined flag, `FRAMEF_SPECIFY`.

The idea behind this frame business is to support multiple gadgets (as found in a system requester) which share a single frame image object, but render it in different dimensions appropriate for their enclosed label. The operation of this method should be defined by suitable subclasses as follows, and return non-zero to indicate that they support this method.

If `FRAMEF_SPECIFY` is not set, you should set up the povided frame box with the position and dimensions of the frame you would render to enclose the contents box. This is an inquiry function used, for example, by a gadget which wants to know what sized frame dimensions to pass to `IM_DRAWFRAME` (defined below) and how to center the contents.

The caller might then add a little aesthetic margin by increasing the dimensions (or might decide to make a column of framed objects all the same width as the largest one), and having done so, can call the method `IM_FRAMEBOX` again, this time with `FRAMEF_SPECIFY` set. Now, your method is expected to respect the dimensions specified in the frame box (even if you think they are too small!) and to set up only the position of the frame box appropriately for enclosing the provided contents box. Currently implemented frame image classes always center the contents, but future classes might want to support an attribute for optional left or right justification of the contents.

`IM_DRAWFRAME` - Image subclasses which assign proper meaning to this class (i.e. *frameiclass*) should respect the dimensions that are passed in this message by superceding this method. If *imageclass* sees this message, it will convert it to an `IM_DRAW` message and send it back to the image’s true class. This way, any subclass of *imageclass* which doesn’t implement or recognize the `IM_DRAWFRAME` method will default to drawing objects at their "natural" dimensions.

`IM_HITFRAME` - Perform a hit test for the image respecting the frame dimensions passed in the message. Useful for something like a stretchable rounded box. This class just performs the same as `IM_HITTEST`, ignoring the dimensions passed.

`IM_ERASEFRAME` - Erase an image respecting the frame dimensions passed in the message. This class calls EraseRect() for the frame dimensions, again not checking dimension validity.

Changed Methods:

`OM_NEW` - Instance data contains an Image structure, and its Depth field is initialized to `CUSTOMIMAGEDEPTH`, which identifies such images to Intuition. The Width and Height fields are set to arbitrary positive numbers for safety, but you should always set them to something meaningful.

`OM_SET` - Applies all supported attributes, returns '1'.

Attributes:

`IA_Left` (ISG)
`IA_Top` (ISG)
`IA_Width` (ISG)
`IA_Height` (ISG)
These attributes are stored (by *imageclass*) as values in the familiar Image structure.

`IA_FGPen` (ISG)
`IA_BGPen` (ISG)
These attributes are maintained in the PlanePick and PlaneOnOff fields, which concept they generalize.

`IA_Data` (ISG)
A pointer to general image data. This value is stored in the ImageData field of the old Image structure.

`IA_LineWidth` ()
A no-op for *imageclass*.

`IA_Pens` ()
Defined but not supported by *imageclass*. Pointer to UWORD pens[], terminated by ~0. This can be used to the exclusion of DrawInfo attributes or parameters, or as an override.

Class: ***frameiclass***
Superclass: *imageclass*
Description: A raised, sunken, or chisled image class supporting frame dimensions and Intuition new look parameters.
Include File: *<intuition/imageclass.h>*

New Methods:

None, but implements some of the no-ops in *imageclass*.

Changed Methods:

`OM_GET` - A no-op for this class.

`IM_DRAW` - Draws the embossed frame in its natural (Image) dimensions. Frame thickness support of `IA_LineWidth` is currently not implemented. Frame type is determined by attributes below, colors determined by DrawInfo and supports these drawing states:

`IDS_NORMAL, IDS_INACTIVENORMAL, IDS_DISABLED:`
Renders edges in shadowPen, shinePen, and backgroundPen for the interior. NOTE: No ghosting is done for `IDS_DISABLED.`

`IDS_SELECTED, IDS_INACTIVESELECTED:`
Renders edges in shadowPen, shinePen, and hifillPen, for the interior.

`IM_DRAWFRAME` - Same as `IM_DRAW`, but draws frame to specified exterior dimensions.

`IM_FRAMEBOX` - Centers contents, adds dimension for frame thickness, plus an interior margin of the same thickness.

Attributes:

`IA_Recessed` (IS)
Specifies that the frame should be recessed into the drawing surface (shadowPen used on upper and left edges). Default is `FALSE`, a raised frame.

`IA_DoubleEmboss` (IS)
Two nested embossed frames for "chisled" boxes or "ridges". Not currently supported.

`IA_EdgesOnly` (IS)
Does not fill frame, just draws the edges.

`IA_LineWidth` ()
Not currently supported.

Class: ***sysiclass***
Superclass: *imageclass*
Description: Class for system and standard application images.
Include File: *<intuition/imageclass.h>*

New Methods:

None.

Changed Methods:

`OM_NEW` - Executes a drawing commands list to create a fast-rendering image object. The glyph of the object is defined by the attribute `SYSIA_Which`. The `SYSIA_DrawInfo` parameter is *required* at initialization, as is `IA_Height`. Image size defaults are established by `SYSIA_Size`, but some images (title bar gadget images) require `IA_Height`. Images are scaled to their dimensions.

Attributes:

`SYSIA_DrawInfo` (IS)
This attribute must be passed to `OM_NEW` and `OM_SET` to allow the image to be generated into a bitmap cache.

`SYSIA_Which` (I)
Identifies which of the system image glyphs the caller wants. It can be one of the following:

`DEPTHIMAGE`	Window depth arrangement image.
`ZOOMIMAGE`	Window Zoom image.
`SIZEIMAGE`	Window Sizing image.
`CLOSEIMAGE`	Window close image.
`SDEPTHIMAGE`	Screen depth arrangement image.
`LEFTIMAGE`	Left arrow image.
`UPIMAGE`	Up arrow image.
`RIGHTIMAGE`	Right arrow image.
`DOWNIMAGE`	Down arrow image.
`CHECKIMAGE`	Checkmark image.
`MXIMAGE`	Radio button image.

`SYSIA_Size` (I)
Identifies which default dimensions to use for the object. This generalizes Intuition's older concept of two different system image dimensions. The system currently uses `SYSISIZE_MEDRES` as the default, `SYSISIZE_HIRES` for screens with title bars greater than 22 pixels, and `SYSISIZE_LORES` for screens with fat pixels.

Class: ***fillrectclass***
Superclass: *imageclass*
Description: Rectangle with frame and pattern support
Include File: *<intuition/imageclass.h>*

New Methods:

None.

Changed Methods:

`IM_DRAW` - Draws the pattern in its natural (Image) dimensions.

`IM_DRAWFRAME` - Same as `IM_DRAW`, but draws pattern to specified exterior dimensions.

Attributes:

`IA_APattern` (IS)
Points to the pattern to fill in the Image structure's ImageData field.

`IA_APatSize` (IS)
The pattern's dimension.

`IA_Mode` (IS)
Drawing mode for the pattern.

Class: ***itexticlass***
Superclass: *imageclass*
Description: IntuiText equivalent with attribute override.
Include File: *<intuition/imageclass.h>*

New Methods:

None.

Changed Methods:

`IM_DRAW` - Draws IntuiText specified as `IA_Data` overriding the attributes specified in the IntuiText structure. The Mode used is JAM1, the color as specified as `IA_FGPen`. The position of the itexticlass object is added to the position of the IntuiText.

This class was defined to make new-looking AutoRequesters without changing the IntuiText structures passed to it.

Attributes:

`IA_Data` (ISG)
Processed blindly by superclass, must be a pointer to a normal IntuiText structure.

`IA_FGPen` (ISG)
Processed by superclass, used as APen when drawing text.

`IA_Left`, `IA_Top` (ISG)
Added to IntuiText offsets.

Class: ***icclass***
Superclass: *rootclass*
Description: Base class of simple forwarding interconnection.
Include File: *<intuition/icclass.h>*

New Methods:

`ICM_SETLOOP`
`ICM_CLEARLOOP`
`ICM_CHECKLOOP` - All of these methods are used internally by subclasses to manage a loop inhibition for broadcasted messages. They increment, decrement, and return the value of that counter.

Changed Methods:

`OM_SET` - Sets attributes and returns 0. Note that this is *not* the same behavior as `OM_NOTIFY`.

`OM_NOTIFY` - Issues an `OM_UPDATE` message to the object indicated by attribute `ICA_Target`, first converting the attribute tags according to the `ICA_Map` attribute. The internal loop count is incremented until the `OM_UPDATE` method it invokes on `ICA_Target` returns. Also, it will do nothing at all if that loop count was non-zero to begin with, preventing itself from participating in an infinite loop.

`OM_UPDATE` - Receive update notices from others. This base IC class just passes the message along, treating it exactly like an `OM_NOTIFY` message. Subclasses will probably redefine this method to perform a calculation, the result of which they might forward to others by issuing themselves an `OM_NOTIFY` message. Unfortunately, this class *converts* the `OM_SET` and `OM_NOTIFY` messages to an `OM_NOTIFY` by changing their MethodID field, and does not restore it to what it originally was.

Attributes:

`ICA_Target` (IS)
Target object for `OM_UPDATE` messages. If this attribute is given the value `ICTARGET_IDCMP`, then the notification will consist of sending an IDCMPUPDATE IntuiMessage to a window. The window is determined by the GadgetInfo structure passed around when the object is connected to gadgets.

`ICA_Map` (IS)
Attribute mapping list.

`ICSPECIAL_CODE` (*)
This is a magic dummy attribute: if it occurs as a target in the `ICA_Map` mapping list, and `ICA_Target` is `ICTARGET_IDCMP`, then the new value of the corresponding attribute will be copied into the IntuiMessage.Code field of the IDCMPUPDATE message (just the lower sixteen bits of the attribute value will fit). This sometimes makes it particularly simple to process IDCMPUPDATE messages with a single item of interest.

Class: ***modelclass***
Superclass: *icclass*
Description: Provides "broadcast" interconnection.
Include File: *<intuition/icclass.h>*

New Methods:

None.

Changed Methods:

`OM_ADDMEMBER` - Adds an object to the the Model's internally maintained broadcast list. *Anything on this list will be DISPOSED when the model object is disposed.*

`OM_REMMEMBER` - Removes objects added by `OM_ADDMEMBER`.

`OM_DISPOSE` - Disposes members of the broadcast list as well as itself.

`OM_NOTIFY`, `OM_UPDATE` - these behave like the *icclass* methods, but first an `OM_UPDATE` message is sent to all members of the object's broadcast list, un-mapped. The members of the broadcast list are typically icclass objects, which have gadgets as their targets and appropriate mapping lists. If you are creating a useful subclass, you will probably want to intercept `OM_UPDATE`, but pass `OM_NOTIFY` to *modelclass*, your superclass.

Attributes:

`ICA_Map`, `ICA_Target` - Same as for the superclass (*icclass*).

Class:	***gadgetclass***
Superclass:	*rootclass*
Description:	Base class for Intuition compatible gadget classes
Include File:	*<intuition/gadgetclass.h>*

New Methods:

Intuition uses the following methods to control a gadget. Only custom gadget class implentors will be interested in these.

`GM_HITTEST`
`GM_RENDER`
`GM_GOACTIVE`
`GM_HANDLEINPUT`
`GM_GOINACTIVE`

Changed Methods:

`OM_NEW` - Initializes transparent Gadget structure. LeftEdge and TopEdge are set to 0, Width and Height are arbitrary constants which you should override. Sets GadgetType to CUSTOMGADGET, and installs a pointer to the *gadgetclass* data structure in the MutualExclude field. These objects are also valid "custom gadgets."

Links self into a NextGadget linked list if `GA_Previous` is passed to `OM_NEW`.

`OM_UPDATE` and `OM_SET` are used simply to change the attributes. This *gadgetclass* does not implement any concrete gadgets. This is left to the subclasses.

`OM_NOTIFY` - This class will issue an `OM_UPDATE` message to `ICA_Target` through `ICA_Map`. Subclasses of *gadgetclass* are advised to let the superclass handle this method.

Attributes:

`GA_Left` (IS)
`GA_Top` (IS)
`GA_Width` (IS)
`GA_Height` (IS)
These correspond to position and dimension fields in the Intuition Gadget structure. Setting these *clears* the "gadget relative" flags (below).

`GA_RelRight` (IS)
`GA_RelBottom` (IS)
`GA_RelWidth` (IS)
`GA_RelHeight` (IS)
These are alternative position/dimension attributes. Setting these stores the corresponding data in the Left/Top/Width/Height Gadget fields, respectively, and sets the corresponding "relative" flag (`GRELRIGHT`, `GRELBOTTOM`, and so on).

`GA_IntuiText` (IS)
`GA_Text` (IS)
`GA_LabelImage` (IS)
This copies the ti_Data value blindly into the Gadget.GadgetText field, and sets the flags `LABELSTRING` and `LABELIMAGE` as appropriate in Gadget.Flags. `GA_IntuiText` requires that ti_Data be an IntuiText pointer, as with old-style gadgets. `GA_Text` takes a pointer to a `NULL`-terminated string (UBYTE *). `GA_LabelImage` takes a pointer to a (boopsi) image.

Classes which support the attributes other than `GA_IntuiText` must document themselves appropriately, but this facility allows us to have low-overhead text gadgets and gadgets with iconic labels.

`GA_Image` (IS)
This may be a boopsi image or a regular image. Things are designed so that the image may be shared between gadgets. The image will *not* be disposed when the gadget object is disposed.

`GA_Border` (IS)
`GA_SelectRender` (IS)
`GA_ID` (IS)
`GA_UserData` (IS)
`GA_SpecialInfo` (IS)
All of this group of attributes correspond to fields in the Gadget structure.

`GA_Disabled` (IS)
`GA_GZZGadget` (IS)
`GA_Selected` (IS)
`GA_EndGadget` (IS)
`GA_Immediate` (IS)
`GA_RelVerify` (IS)
`GA_FollowMouse` (IS)
`GA_RightBorder` (IS)
`GA_LeftBorder` (IS)
`GA_TopBorder` (IS)
`GA_BottomBorder` (IS)
`GA_ToggleSelect` (IS)
`GA_SysGadget` (IS)
`GA_TabCycle` (IS)
This group corresponds to Boolean attributes (flags) in the Gadget structure. This class will put the correct flag in the correct field. You can pass any non-zero value in the ti_Data tag field to cause the corresponding flag to be set. Pass zero to clear.

`GA_HighLight` (IS)
Sets the "`GADGHIGHBITS`" portion of Gadget.Flags to the attribute value in ti_Data.

`GA_SysGType` (IS)
Sets the system gadget type portion of Gadget.GadgetType to the (masked) value in ti_Data.

`GA_Previous` (I)
Previous gadget (or (struct Gadget **)) in linked list. This attribute cannot be used to link new gadgets into the gadget list of an open window or requester. You must use AddGList().

`ICA_Target` (IS)
`ICA_Map` (IS)
These are given the same meaning as in *icclass*, from which they are borrowed.

Class: ***propgclass***
Superclass: *gadgetclass*
Description: Extended function proportional gadgets.
Include File: *<intuition/gadgetclass.h>*

New Methods:

None.

Changed Methods:

All methods defined by *gadgetclass* are handled to provide compatible proportional gadget processing.

`OM_SET`, `OM_UPDATE` - Changes attributes and, if needed and if propgclass is the true class, will update the slider visuals by updating the knob position and dimensions.

`GM_HANDLEINPUT` - If the knob position changes sufficiently to change `PGA_Top`, this method will issue an `OM_NOTIFY` message, with attributes `PGA_Top` and `GA_ID`. The `OPUF_INTERIM` flag will be set for intermediate messages issued while the mouse is dragging the slider knob. This method Will issue a message with `OPUF_INTERIM` clear when it is done, which is the "final value".

Attributes:

`GA_Image` (I)
`GA_Border` (I)
If you don't pass `GA_Image` to NewObject(), the gadget will create and use an AUTOKNOB. Likewise if you pass (the "illegal" attribute) `GA_Border`: an AUTOKNOB will be used instead.

`GA_HighLight` (I)
GADGHBOX highlighting is not allowed, and will be converted to GADGHBOX.

`GA_SpecialInfo` ()
This attributed is "forced" to point to the PropInfo allocated for objects of this class.

Other *gadgetclass* attributes are passed along to the superclass.

`PGA_Freedom` (IG)
May be either of `FREEHORIZ` or `FREEVERT` (not both). The default is `FREEVERT`.

`PGA_Borderless` (I)
Means the same as PropInfo.Flags `BORDERLESS`.

`PGA_HorizPot, PGA_VertPot, PGA_HorizBody, PGA_VertBody`
These are defined in the include file but obsolete and will not be supported. Class propgclass supports gadgets that are vertical or horizontally free, but not both.

`PGA_Top` (ISGNU)
`PGA_Visible` (ISU)
`PGA_Total` (ISU)
These attributes are very useful replacements to Pot and Body variables. They are based on the use of proportional gadgets to control scrolling text. When scrolling 100 lines of text in a 25 line visible window, you would set `PGA_Total` to 100, `PGA_Visible` to 25, and watch `PGA_Top` run from 0 to 75 (the top line of the last page). All internal prop gadget values will be calculated based on these (depending on whether the gadget is FREEHORIZ or FREEVERT). "Container clicks" for page jumps will leave an overlap of one line, unless the value `PGA_Visible` is 1, in which case the prop gadget acts as an integer numeric slider taking values from 0 to `PGA_Total` - 1. Note that `PGA_Top` has `OM_NOTIFY` access. All three of these attributes have `OM_UPDATE` access, so they can be controlled via interconnections.

`PGA_NewLook` (I)
Equivalent to PropInfo.Flags `PROPNEWLOOK` flag.

Class: ***strgclass***
Superclass: *gadgetclass*
Description: Intuition compatible string gadgets.
Include File: *<intuition/gadgetclass.h>*

New Methods:

None.

Changed Methods:

All methods defined by *gadgetclass* are handled to provide compatible string gadget processing.

`OM_NEW` - Sets up StringInfo and StringExtend structures. Will allocate a Buffer if needed and will use shared data buffers for UndoBuffer and WorkBuffer if the MaxChars is less than `SG_DEFAULTMAXCHARS` (128). Default text pens are FG = 1, BG = 0;

Attributes:

`GA_ID` (ISG)
Will be included in `OM_NOTIFY` messages generated.

`STRINGA_MaxChars` (I)
`STRINGA_Buffer` (I)
`STRINGA_UndoBuffer` (I)
`STRINGA_WorkBuffer` (I)
Specify various buffers defined for string gadgets and extended string gadgets. If your value of `STRINGA_MaxChars` is less than `SG_DEFAULTMAXCHARS` (120 for now), then this class can provide all these buffers for you. Note that UndoBuffer and WorkBuffer can be shared by many separate gadgets, providing they are as large as the largest MaxChars they will encounter.

`STRINGA_BufferPos` (ISU)
`STRINGA_DispPos` (ISU)
Familiar cursor and scroll position.

`STRINGA_AltKeyMap` (IS)
Same as StringInfo.AltKeyMap.

`STRINGA_Font` (IS)
Font for string gadget text. Must be an open TextFont.

`STRINGA_Pens` (IS)
Pen numbers, packed as two shorts into a longword, for rendering gadget text.

`STRINGA_ActivePens` (IS)
Optional pen numbers, packed as two shorts into a longword, for rendering gadget text, when the gadget is active.

`STRINGA_EditHook` (I)
Custom string gadget edit hook.

`STRINGA_EditModes` (IS)
Value taken from flags defined in *<intuition/sghooks.h>* for initial editing modes.

`STRINGA_ReplaceMode` (IS)
`STRINGA_FixedFieldMode` (IS)
`STRINGA_NoFilterMode` (IS)
These three are independent Boolean equivalents to the individual flags that you can set for `STRINGA_EditModes`.

STRINGA_Justification (IS)
Takes the values STRINGCENTER, STRINGRIGHT, and STRINGLEFT (which is 0).

STRINGA_LongVal (ISGNU)
When you specify this to a string gadget object, it means first that the gadget is now for integer entry only, and the value of the gadget takes the numeric value passed in ti_Data. Note that this attribute has OM_NOTIFY and OM_UPDATE access.

STRINGA_TextVal (ISGNU)
When you specify this to a string gadget object, it means that the gadget is for text entry only, and the value of the gadget is copied from the string value passed as a UBYTE pointer in ti_Data. Note that this attribute has OM_NOTIFY and OM_UPDATE access.

STRINGA_ExitHelp (IS)
Set this if you want the gadget to exit when the ‘‘Help’’ key is pressed. Look for a code of 0x5f, the rawkey code for help.

Notification messages will be issued whenever the gadget chooses to go inactive (not when it is aborted). The OPUF_INTERIM flag is always clear.

Class:	***groupgclass***
Superclass:	*gadgetclass*
Description:	Composite gadget objects
Include File:	*<intuition/gadgetclass.h>*

New Methods:

None

Changed Methods:

OM_SET - Passes most attributes to the superclass, but propagates changes in position to its members appropriately. Also, GA_Width and GA_Height are calculated from the position and dimension of the membership.

OM_ADDMEMBER - add a gadget to the group list. Add the gadgets you want in the group right after you create it and leave them there until you are done. Note that all members of the *groupgclass* object will be deleted by OM_DISPOSE.

OM_DISPOSE - this class will dispose of the object and all its member gadgets.

Attributes:

LAYOUTA_Orientation
LAYOUTA_Spacing
LAYOUTA_LayoutObj
These attributes specify simple layout parameters and a "layout object delegate" are not currently implemented. GA_Width and GA_Height are calculated from the membership.

GA_Left (IS)
GA_Position (IS)
These are propagated magically to the membership.

The gadget relative flags (GA_RelWidth, GA_RelHeight, and so on) are not supported.

Class: ***buttongclass***
Superclass: *gadgetclass*
Description: A (repeatable) command button gadget.
Include File: *<intuition/gadgetclass.h>*

New Methods:

None.

Changed Methods:

GM_HITTEST - Delegates this question to its GA_Image attribute.

GM_HANDLEINPUT - Will behave like a button, but continuously issues OM_NOTIFY messages for each IECLASS_TIMER event.

Flag OPUF_INTERIM will be set for all but the last notification.

The notified attribute is GA_ID, with a twist: the value sent will be the *negative* of GadgetID if the pointer is not currently over the gadget image.

GM_RENDER - All rendering is passed along to the GadgetRender Image. The state provided is IDS_INACTIVESELECTED, IDS_INACTIVENORMAL, IDS_SELECTED, or IDS_NORMAL.

Attributes:

GA_Image (IS)
Changing the image will cause the gadget to refresh itself.

Class: ***frbuttonclass***
Superclass: *buttongclass*
Description: A button gadget that knows how to outline its "label" within a shared "frame image".

New Methods:

None.

Changed Methods:

OM_NEW - Will set up its dimensions depending on GA_Image, including support for frame images. If GA_Image understands the IM_FRAMEBOX method, dimensions are calculated to surround the "label" stashed in GadgetText, which can be GA_IntuiText, GA_Text, or GA_LabelImage.

OM_SET - If you change the dimensions, this method will adjust the contents by using IM_FRAMEBOX with FRAMEIF_SPECIFY.

GM_HITTEST - uses IM_HITFRAME.

GM_RENDER - uses IM_DRAWFRAME. First draws the frame, then draws the contents or label described under OM_NEW.

Attributes:

GA_Width (IS)
GA_Height (IS)
If you change these, the contents will be readjusted and the gadget re-rendered.

❖