Libraries

COLLABORATORS

	TITLE : Libraries	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 14, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Libraries

1.1 Amiga® RKM Libraries: B Boopsi Class Reference

```
                The Boopsi Class Reference documents all of the classes built into
Intuition.  Each class entry in the reference starts off with:

   Class:          The class's name (for example,
                  gadgetclass
                  )
   Superclass:     The class's superclass (for example,
                  rootclass
                  )
   Include File:   The class's include file (for example,
                    <intuition/gadgetclass.h>)

The include file contains the class's message structures, attribute IDs,
and method IDs.  This is followed by a short description of the class.

The rest of a class entry is broken up into three sections:

  * New Methods      Describes each new method that the class defines.

  * Changed Methods  Describes each method to which this class makes
                     significant changes.

  * Attributes       Describes the attributes that this class defines as
                     well as inherited ones that this class alters.
```

gadgetclass

frbuttonclass

icclass

sysiclass

propgclass

groupgclass

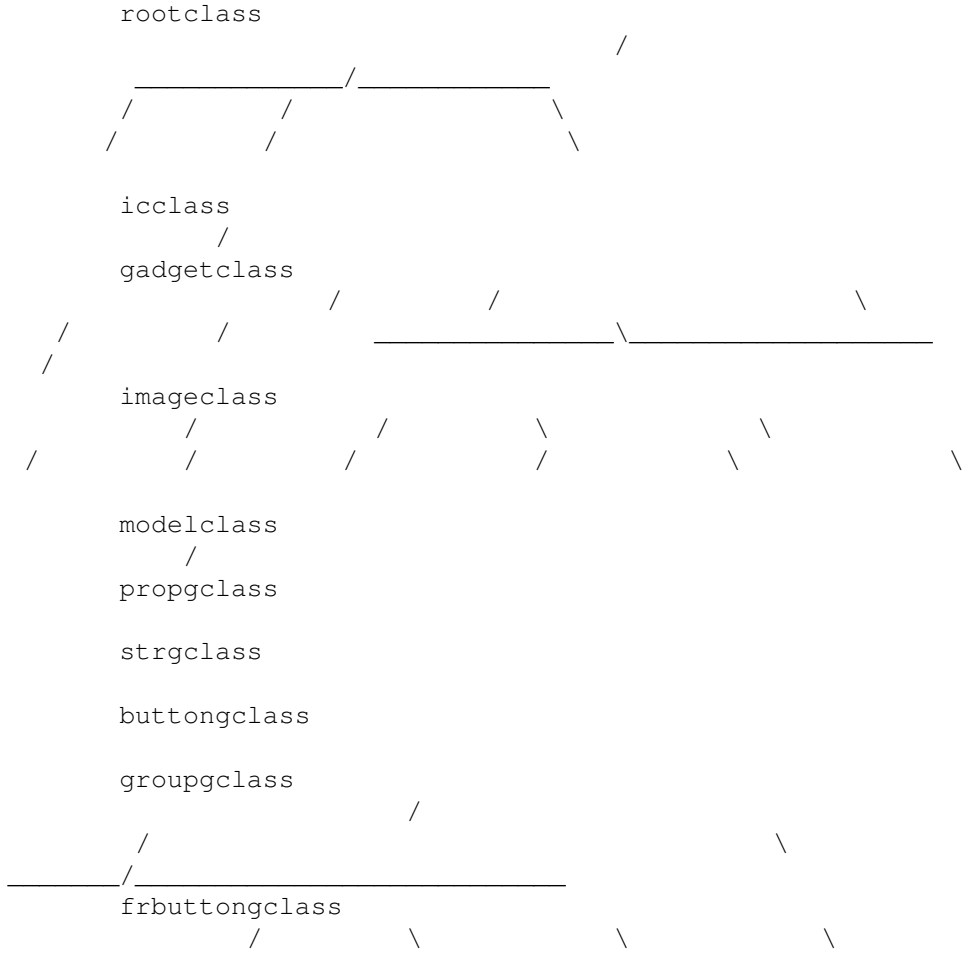
modelclass

fillrectclass

strgclass

1.2 B Boopsi Class Reference / Introduction

There are 14 public classes built into the Release 2.04 ROM:



```
       /           \            \            \

                frameiclass

                sysiclass

                fillrectclass

                itexticlass
                This appendix documents all the standard Boopsi classes, including ↩
                    their
methods and attributes.

Each class entry in this document can have two sets of methods: new
methods that the class defines and inherited methods that the class has
modified significantly.  Similarly, each class entry can have two sets of
attributes: those that the class defines and those that the class
inherited and modified.  Unless documented otherwise,  all classes inherit
all of its superclass's methods and attributes.

Each method has a Boopsi message associated with it.  These messages are
in the form of C structures.  Many methods use the default message
structure:

    typedef struct
    {
        ULONG MethodID;
    } *Msg;

Some methods require a customized message so they can pass other
parameters besides the Method ID.  If a method requires a custom message,
its custom message structure is documented along with the method.

All methods have to return some sort of return value.  The meaning of the
return value depends on the class and method.  Normally a return value of
zero indicates that either the method failed or it is not supported by the
class.  A method can use the return value to return a pointer to an
object.  If a class does not directly support a particular method, the
class's dispatcher should pass the method on to its superclass for
processing.  The class dispatcher should record the return value it gets
from its superclass and use that as its return value.  Methods that assign
no meaning to their return value can return 1L to indicate that the method
is implemented.

The description of each attribute contains a code which lists the

                rootclass
                 methods that apply to that attribute:

    I
                OM_NEW
                     Attribute can be set at initialization
    S
                OM_SET
                     Attribute can be set with OM_SET method
    G
                OM_GET
```

```
                    Attribute can be read with OM_GET method
    N
                OM_NOTIFY
                  Changing the attribute triggers object to send
                  notifications
    U
                OM_UPDATE
                  Attribute can be set with OM_UPDATE method
```

For example, the `itexticlass` attribute `IA_Left` has the code (ISG) after it. This means an application can set IA_Left when it creates an instance of itexticlass (`OM_NEW`) and when it uses the OM_SET method. The application can also ask an itexticlass object what the IA_Left value is, using the `OM_GET` method.

The `OM_NEW`, `OM_SET`, `OM_NOTIFY`, and `OM_UPDATE` messages all contain a pointer to a tag list. This tag list contains the attributes and corresponding values that the method affects. Each TagItem in this list makes up an attribute/value pair. The ti_Tag portion of the TagItem contains the attribute's ID while the ti_Data field contains the attribute's value. Note that these tag lists can also contain utility.library Global System control tags (like TAG_SKIP and TAG_DONE), so dispatchers should use the tag functions from utility.library to process these lists. See documentation on the Utility library for more information.

All methods are called via a class dispatcher:

```
classDispatcher(Class *class, Object *object, Msg msg);
```

The first argument, class, is a pointer to the dispatcher's Class structure (defined in <intuition/classes.h>). The second parameter, object, is a pointer to the Boopsi object to which the Boopsi message (the third argument, msg) refers. Both Object and Msg are defined in <intuition/classusr.h>.

1.3 B Boopsi Class Reference / rootclass

```
                Class:                  rootclass
Superclass:             None
Include File:           <intuition/classusr.h>

This is the universal base class for all other classes.

New Methods:
------------

                 OM_NEW

                 OM_ADDTAIL

                 OM_DISPOSE

                 OM_REMOVE
                The following methods are described at the rootclass level  ↩
                   although its up
to the subclasses to actually implement them.  If a class does not
implement these methods, it should either return zero, indicating that
this class does not support the method, or defer processing on to its
superclass.


                 OM_ADDMEMBER

                 OM_GET

                 OM_UPDATE

                 OM_REMMEMBER

                 OM_SET

                 OM_NOTIFY
                Changed Methods:
----------------
Not applicable.

Attributes:
-----------
None.
```

1.4 B / rootclass / New Methods: OM_NEW

```
                This method tells a class to create a new instance of itself.  If  ↩
                   OM_NEW
is successful, it returns a pointer to the new object, otherwise it
returns NULL.

For programmers who are only creating Boopsi objects rather than creating
custom classes, use the intuition.library function NewObject():
```

```
    APTR NewObject(struct IClass  *privateclass,
                   UBYTE          *publicclassname,
                   ULONG          firsttag,
                   ...)
```

The OM_NEW method receives the following arguments (defined in <intuition/classusr.h>):

```
    struct opSet /* The OM_NEW method uses the same structure as OM_SET */
    {
        ULONG             MethodID;      /* OM_NEW */
        struct TagItem    *ops_AttrList; /* tag attributes to initialize */
        struct GadgetInfo *ops_GInfo;    /* Always NULL for OM_NEW */
    };
```

The ops_AttrList field contains a pointer to a tag list of attribute/value pairs. Each pair contains an attribute ID and the initial value of the corresponding attribute.

The ops_GInfo field is always NULL for the OM_NEW method.

Unlike other methods, when the dispatcher gets an OM_NEW message, the object pointer (newobject from the dispatchRKMModel() prototype above) does not point to an object, since the idea is to create a new object. The pointer normally used to pass a Boopsi object is instead used to pass the address of the object's "true class" (the class of which the object is an instance).

The first thing the dispatcher does when it processes an OM_NEW message is pass the OM_NEW message on to its superclass's dispatcher. It does this using the amiga.lib function DoSuperMethodA():

```
    ULONG DoSuperMethodA(Class *cl, Object *trueclass, Msg msg);
```

Each superclass's dispatcher does this until the message gets to the rootclass dispatcher.

Each class keeps a record of how much memory its local instance data requires. The rootclass dispatcher's OM_NEW method looks at the object's true class (newobject from the prototype) to find out how much memory to allocate for the object's instance data. The rootclass dispatcher allocates enough memory for the true class's local instance data, plus enough memory for the local instance data of each of the true class's superclasses. If all goes well, the rootclass dispatcher increments the true class's internal count of instances of true class, and returns a pointer to the newly created object. it passes control back to the subclass dispatcher that called it. If there was a problem, the rootclass dispatcher passes back a NULL.

When the rootclass dispatcher returns, the subclass dispatcher regains control from DoSuperMethodA(). DoSuperMethodA() will return either a pointer to the new object or NULL if there was an error. Although the rootclass dispatcher allocated all the memory the object needs, it did not

set up any of that memory. Now its the the subclass dispatcher's turn to do some work. It has to initialize the instance data that is local to its class. A dispatcher finds its local instance data by using the INST_DATA() macro (defined in <intuition/classes.h>).

After initializing its local instance data, the subclass dispatcher passes control down to its subclass dispatcher, which in turn, initializes its local instance data. Control passes down from class to subclass until the true class dispatcher regains control.

Now the true class dispatcher has to initialize its local instance data. It has to scan through the tag list of attribute/value pairs passed in the OM_NEW message (opSet.ops_AttrList). If the dispatcher finds any attributes that the true class recognizes, it has to initialize them to the value passed in the attribute/value pair.

At this point, the new object can allocate other resources it needs that it did not allocate as part of its instance data. For example, the new Boopsi object might need a frame image around itself, so it can create a new one using a Boopsi frame image. If the object allocates any resources in this step, it must deallocate these resources later when it is disposed in the

OM_DISPOSE
 method.

Finally, the dispatcher can return. When the dispatcher returns from an OM_NEW method, it returns a pointer to the new object.

1.5 B / rootclass / New Methods: OM_DISPOSE

This method instructs an object to delete itself. The rootclass
dispatcher's OM_DISPOSE method decrements the true class's ↩
internal count
of instances of true class. The return value for this method is not explicitly defined.

This method uses the default Boopsi message.

Applications should not call this method directly. Instead they should use the intuition.library function DisposeObject().

For the OM_DISPOSE method, an object should do the following:

Free any additional resources the object explicitly allocated itself in the

OM_NEW
 method (this does not include the instance data).

Pass the message up to the superclass, which will eventually reach

rootclass
, which will free the instance data allocated for the object.

```
If a class does not allocate any extra resources when it creates an
object, it can defer all OM_DISPOSE processing to its superclass.
```

1.6 B / rootclass / New Methods: OM_ADDTAIL

```
This method tells an object to add itself to the end of a specified Exec
list.  Boopsi objects contain a MinNode structure used for this purpose.
The return value for this method is not explicitly defined.

The method uses a custom message (defined in <intuition/classusr.h>):

    struct opAddTail {
        ULONG       MethodID;   /* OM_ADDTAIL */
        struct List *opat_List; /* The exec list to add the object to */
    };

The opat_List can be any Exec list.  Use the Intuition function
NextObject() to step through this list.
```

1.7 B / rootclass / New Methods: OM_REMOVE

```
Remove an object from an Exec list.  The return value for this method is
not explicitly defined.  This method uses the default Boopsi message.
```

1.8 B / rootclass / New Methods: OM_ADDMEMBER

```
                Tells an object to add another object to its personal Exec list.   ↩
                   What the
list is for depends on the class.  The return value for this method is not
explicitly defined.

One class that uses this method is
                modelclass
                .  A modelclass object
maintains a broadcast list.  When a modelclass object gets an
                OM_NOTIFY
                message, it broadcasts an
                OM_UPDATE
                 message about the OM_NOTIFY to every
object in its broadcast list.

This method uses a custom message (defined in <intuition/classusr.h>):

    #define  opAddMember opMember
    struct opMember {
        ULONG  MethodID;     /* OM_ADDMEMBER (or OM_REMMEMBER) */
        Object *opam_Object; /* add (or remove) this object    */
    };                       /* to (from) personal list.       */
```

```
opam_Object is the object to add to the list.  A dispatcher typically
implements OM_ADDMEMBER by sending the
                OM_ADDTAIL
                 message to the
opam_Object object.
```

1.9 B / rootclass / New Methods: OM_REMMEMBER

```
                Tells an object to remove a member object from its personal list.  ↩
                    The
member object should have already been added with
                OM_ADDMEMBER
                .  This
method uses the same custom message as OM_ADDMEMBER.  Normally a
dispatcher implements OM_REMMEMBER by sending the
                OM_REMOVE
                 message to the
opam_Object object.  The return value for this method is not explicitly
defined.
```

1.10 B / rootclass / New Methods: OM_GET

```
Tells an object to report an attribute's value.  Applications should not
call this method directly.  Instead, use the intuition.library function
GetAttr().  The return value for this method is not explicitly defined.

This method uses a custom message (defined in <intuition/classusr.h>):

    struct opGet {
        ULONG MethodID;     /* OM_GET */
        ULONG opg_AttrID;   /* ID of attribute to get */
        ULONG *opg_Storage; /* place to put attribute value */
    };

If the object's dispatcher recognizes opg_AttrID as one of the attributes
defined by this class, the dispatcher should copy the value of that
attribute to where opg_Storage points:

    struct opGet *myopget;
    ...
    *(myopget->opg_Storage) = my_attribute_value;
    ...

If the dispatcher does not recognize opg_AttrID, it should pass the
message on to the superclass.
```

1.11 B / rootclass / New Methods: OM_SET

This method tells an object to set one or more of its attributes. Applications should not call this method directly. Instead, use the intuition.library functions SetAttrs() and SetGadgetAttrs() to call this method. The return value for this method is not explicitly defined.

The return value for this method is not explicitly defined. However, in general, when implementing the OM_SET method, if setting an object attribute causes some sort of visual state change, the OM_SET method should return a value greater than zero. If changing an attribute does not affect the visual state, OM_SET should return zero.

This method uses a custom message (defined in <intuition/classusr.h>):

```
struct opSet {
    ULONG             MethodID;      /* OM_SET */
    struct TagItem    *ops_AttrList; /* tag list of attributes to set*/
    struct GadgetInfo *ops_GInfo;
};
```

The ops_AttrList field contains a pointer to a tag list of attribute/value pairs. These pairs contain the IDs and the new values of the attributes to set. The dispatcher has to look through this list (see docs for the utility.library NextTagItem() function) for attributes its class recognizes and set the attribute's value accordingly. The dispatcher should let its superclass handle any attributes it does not recognize.

If the object is a gadget, the ops_GInfo field contains a pointer to a GadgetInfo structure. Otherwise, the value in ops_GInfo is undefined. Intuition use the GadgetInfo structure to pass display information to gadgets. See the

gadgetclass

methods for more details.

1.12 B / rootclass / New Methods: OM_UPDATE

This method tells an object to update one or more of its attributes. No application should call this method. Only Boopsi objects send OM_UPDATE messages. The return value for this method is not explicitly defined.

A Boopsi object uses an OM_UPDATE message to notify another Boopsi object about transitory changes to one or more attributes.

From the point of view of most objects, an OM_UPDATE message is almost identical to

OM_SET

. Because the methods are so similar, When a typical dispatcher receives an OM_UPDATE message, it processes the OM_UPDATE the same way it would process an OM_SET message, usually using the same code.

There are actually two kinds of OM_UPDATE, an interim and final one. While a Boopsi object's attribute is in a transient state, it can send out interim OM_UPDATE messages to its target(s). For example, while the user

```
is sliding a Boopsi prop gadget, the prop gadget sends interim OM_UPDATE
message about changes to its
                PGA_Top
                 value (the integer value of the prop
gadget is the PGA_Top attribute) to some target object.  When the user
lets go of the prop gadget, the gadget is no longer in a transient state,
so the gadget sends out a final OM_UPDATE about its PGA_Top attribute.
The target object can choose to change one of its attributes based on the
OM_UPDATE messages it receives.

The layout of the OM_UPDATE message is almost identical to the OM_SET
message:

    struct opUpdate {  /* the OM_NOTIFY method also uses this structure */
        ULONG             MethodID;           /* OM_UPDATE               */
        struct TagItem    *opu_AttrList;      /* tag list of attributes */
        struct GadgetInfo *opu_GInfo;         /*   that changed.        */
        ULONG             opu_Flags;          /* The extra field         */
    };

    #define OPUF_INTERIM    (1<<0)

Some dispatchers need to know the difference between an interim and final
OM_UPDATE.  A dispatcher can tell the difference between an interim and
final OM_UPDATE message because the OM_UPDATE message has an extra field
for flags.  If the low order bit (the OPUF_INTERIM bit) is set, this is an
interim OM_UPDATE message.  The interim flag is useful to a class that
wants to ignore any interim messages, processing only final attribute
values.
```

1.13 B / rootclass / New Methods: OM_NOTIFY

```
                This method tells an object to broadcast an attribute change to a  ↩
                   set of
target objects using
                OM_UPDATE
                 messages.  The return value for this method
is not explicitly defined.

The OM_NOTIFY method uses the same message structure as
                OM_UPDATE
                .

Most dispatchers do not handle the OM_NOTIFY message directly.  Normally
they inherit this method from a superclass, so they pass the OM_NOTIFY
message on to the superclass dispatcher.

Although most dispatchers don't have to process OM_NOTIFY messages, most
do have to send them.  Whenever an object receives an
                OM_SET
                 or
                OM_UPDATE
                about one of its attributes, it may need to notify other objects  ↩
                   of the
```

```
change.  For example, when a prop gadget's
                PGA_Top
                 value changes, its
target object(s) need to hear about it.

If an object needs to notify other objects about a change to one or more
of its attributes, it sends itself an OM_NOTIFY message.  The OM_NOTIFY
message will eventually end up in the hands of a superclass that
understands OM_NOTIFY and it will send
                OM_UPDATE
                 messages to the target
objects.
```

1.14 B Boopsi Class Reference / icclass

```
                Class:                  icclass (interconnection class)
Superclass:
                rootclass
                Include File:           <intuition/icclass.h>

Base class of simple OM_UPDATE forwarding objects.  When an icclass object
gets an
                OM_UPDATE
                 message, it maps the attributes in the OM_UPDATE message
according to its mapping list (its
                ICA_MAP
                 attribute) and forwards the

                OM_UPDATE
                 to its target (its
                ICA_TARGET
                 attribute).

New Methods:
------------
None.

Changed Methods:
----------------

                 OM_SET

                 OM_UPDATE/OM_NOTIFY
                Attributes:
-----------

                 ICA_TARGET (IS)

                 ICA_MAP (IS)

                 ICSPECIAL_CODE (*)
```

1.15 B / icclass / Changed Methods: OM_SET

This method sets its attributes and returns 0.

1.16 B / icclass / Changed Methods: OM_UPDATE/OM_NOTIFY

These methods tell the object to notify its
ICA_TARGET
 of attribute
changes by sending the target an OM_UPDATE message. If the object has an

ICA_MAP
, it maps the attribute IDs it finds to new attribute IDs. See ↩
 the

rootclass
 descriptions of
OM_NOTIFY
 and
OM_UPDATE
 for more information.
The return value for this method is not explicitly defined.

1.17 B / icclass / Attributes: ICA_TARGET (IS)

This attribute stores the address of the
icclass
 object's target object.
Whenever the icclass object receives an
OM_NOTIFY
 or
OM_UPDATE
 message, it
forwards that message to its target in the form of an OM_UPDATE message.
If the icclass object has an attribute mapping list (see the
ICA_MAP
attribute below), it also maps the
OM_NOTIFY/OM_UPDATE
 message's attribute
IDs to new ones before forwarding the message.

If the value of ICA_TARGET is ICTARGET_IDCMP, the
icclass
 object sends an
IDCMP_IDCMPUPDATE IntuiMessage to its window instead of forwarding an
OM_UPDATE message. See the
rootclass
 description of
OM_UPDATE
 for more
information.

1.18 B / icclass / Attributes: ICA_MAP (IS)

```
                This attribute points to a tag list of attribute mappings which ↩
                   the

                icclass
                 object uses to change the attribute IDs of an
                OM_UPDATE
                's
attribute/value pairs.  For example, if an icclass object had the
following ICA_MAP:

    struct TagItem map[] =
    {
        {PGA_Top, STRINGA_LongVal},
        {MYATTR, MYNEWATTR},
        {TAG_END, }
    };

before sending an
                OM_UPDATE
                 to its
                ICA_TARGET
                , the
                icclass
                 object scans
through the
                OM_UPDATE
                 message's attribute/value pairs looking for the

                PGA_Top
                 and MYATTR attributes.  If it finds the PGA_Top attribute, it
changes PGA_Top to
                STRINGA_LongVal
                .  Likewise, if the icclass object finds
the MYATTR attribute, it changes MYATTR to MYNEWATTR.  The icclass object
does not disturb the attribute's value.
```

1.19 B / icclass / Attributes: ICSPECIAL_CODE (*)

```
                This is a dummy attribute for the
                ICA_MAP
                .  If any attribute maps to
ICSPECIAL_CODE and
                ICA_TARGET
                 is ICTARGET_IDCMP, then the value of the
mapped attribute will be copied into the IntuiMessage.Code field of the
IDCMP_IDCMPUPDATE message (just the lower sixteen bits of the attribute
value will fit).
```

1.20 B Boopsi Class Reference / modelclass

```
                Class:                  modelclass
Superclass:
                icclass
                Include File:           <intuition/icclass.h>

A class of OM_UPDATE forwarding objects that have multiple targets.  In
addition to the features the modelclass object inherits from
                icclass
                , when
a modelclass object gets an
                OM_UPDATE
                 message, it forwards that
                OM_UPDATE
                message to all of the objects in its broadcast list.

New Methods:
------------
None.

Changed Methods:
----------------

                 OM_ADDMEMBER

                 OM_REMMEMBER

                 OM_DISPOSE

                 OM_NOTIFY/OM_UPDATE
                Attributes:
-----------
None.
```

1.21 B / modelclass / Changed Methods: OM_ADDMEMBER

```
                This method tells a model to add an object to its broadcast list.  ↩
                    When
the object disposes of itself, it will also dispose of any objects
remaining on its broadcast list.  The return value for this method is not
explicitly defined.  See the
                rootclass
                 description of
                OM_ADDMEMBER
                 for
more information.
```

1.22 B / modelclass / Changed Methods: OM_REMMEMBER

```
                This method tells a model to remove an object from its broadcast  ↩
                   list.
The return value for this method is not explicitly defined.  See the

                rootclass
                 description of
                OM_REMMEMBER
                 for more information.
```

1.23 B / modelclass / Changed Methods: OM_DISPOSE

```
This method tells a model to dispose of itself plus the objects remaining
on its broadcast list.  The return value for this method is not explicitly
defined.
```

1.24 B / modelclass / Changed Methods: OM_NOTIFY/OM_UPDATE

```
                This method tells an object to forward the message in the form of  ↩
                   an
OM_UPDATE message to all the objects in its broadcast list.  The

                modelclass
                 does not map the attributes in these OM_UPDATE messages.
Because modelclass inherits behavior from
                icclass
                , if the model has an

                ICA_TARGET
                 and
                ICA_MAP
                , it will also send a mapped OM_UPDATE message to
its ICA_TARGET.  The return values for these methods are not explicitly
defined.  See the
                rootclass

                OM_NOTIFY
                /
                OM_UPDATE
                 and icclass

                OM_NOTIFY/OM_UPDATE
                 descriptions for more information.
```

1.25 B Boopsi Class Reference / imageclass

```
                Class:                  imageclass
Superclass:
                rootclass
                Include File:           <intuition/imageclass.h>

This class is the base class for Boopsi Images.  These images are
backwards compatible with the conventional Intuition Images.  Every Boopsi
image has an Intuition Image structure embedded in it so Intuition can
access the Boopsi image as a conventional Image structure when necessary.
Normally there are no direct instances of this class, only instances of
subclasses of imageclass.

New Methods:
------------

                 IM_DRAW

                 IM_ERASE

                 IM_HITFRAME

                 IM_FRAMEBOX

                 IM_HITTEST

                 IM_DRAWFRAME

                 IM_ERASEFRAME
                Changed Methods:
-----------------

                 OM_NEW

                 OM_SET
                Attributes:
------------

                 IA_Left, IA_Top, IA_Width, IA_Height (ISG)

                 IA_FGPen, IA_BGPen (ISG)

                 IA_Data (ISG)

                 IA_Pens ()
```

1.26 B / imageclass / New Methods: IM_DRAW

```
                This method tells an image object to draw itself.  Applications ↩
                   should not
call this method directly, instead use the intuition.library function
DrawImageState().  The return value for this method is not explicitly
defined.
```

The IM_DRAW method uses a custom message structure:

```
    struct impDraw
    {
        ULONG           MethodID;    /* IM_DRAW                    */
        struct RastPort *imp_RPort;  /* RastPort to render into    */
        struct
        {                            /* X and Y offset relative to */
            WORD X;                  /* the image's IA_Left and    */
            WORD Y;                  /* IA_Top attributes          */
        } imp_Offset;
        ULONG           imp_State;   /* Visual state of image      */
        struct DrawInfo *imp_DrInfo; /* describing rendering area  */
    };
n
```

The imp_State field contains the visual state to render the image. The visual states (defined in <intuition/imageclass.h>) are:

IDS_NORMAL	Render using normal imagery. This is the only kind of imagery available to non-Boopsi images.
IDS_SELECTED	Render using "selected" imagery. "Selected" refers to the state of a gadget's imagery when it is the selected gadget.
IDS_DISABLED	Render using "disabled" imagery. "Disabled" refers to the state of a gadget's imagery when it is disabled. Typically, a disabled image has a ghosting pattern on top of it.
IDS_INACTIVENORMAL	This is a special version of IDS_NORMAL for a "normal" image that is in the border of an inactive window.
IDS_INACTIVESELECTED	This is a special version of IDS_SELECTED for a "selected" image that is in the border of an inactive window.
IDS_INACTIVEDISABLED	This is a special version of IDS_DISABLED for a "disabled" image that is in the border of an inactive window.
IDS_BUSY	Render using "busy" imagery as if the object was the image of a gadget in a busy state. The busy gadget state is not yet supported by Intuition.
IDS_INDETERMINATE	Render using "indeterminate" imagery as if the object was the image of a gadget in an indeterminate state. The indeterminate gadget state is not yet supported by Intuition.

Most image objects do not have different visual states for each possible imp_State. See the

imageclass

entries in this index for more details.

When setting the pens to render an image, use the values from the imp_DrInfo->dri_Pens pen array (Note that it is possible for imp_DrInfo to be NULL). The possible pen values are defined in <intuition/screens.h>. See the "Intuition Screens" chapter of the Amiga ROM Kernel Reference Manual: Libraries for more information on the pen array.

1.27 B / imageclass / New Methods: IM_HITTEST

This method returns TRUE if a point is within the old Image structure box defined by the Image structure's LeftEdge, TopEdge, Width, and Height fields. Subclasses of imageclass can redefine this method if they need to change the criteria for deciding if a point is within an image. Application programs should not call this method directly, instead use the Intuition function PointInImage(). The IM_HITTEST method uses a custom message structure:

```
struct impHitTest
{
    ULONG MethodID; /* IM_HITTEST */
    struct
    {
        WORD X;     /* Coordinates of point to test for hit */
        WORD Y;
    } imp_Point;
};
```

If an image object doesn't need to make any changes to how its superclass handles IM_HITTEST, it can blindly pass this method on to its superclass.

1.28 B / imageclass / New Methods: IM_ERASE

The IM_ERASE method tells an image to erase itself. Applications should not call this method directly, instead they should call the Intuition function EraseImage(). The return value for this method is not explicitly defined.

The IM_ERASE method uses a custom message structure:

```
struct impErase
{
    ULONG           MethodID;   /* IM_ERASE */
    struct RastPort *imp_RPort; /* The image's RastPort */
    struct
    {
```

```
            WORD X;                 /* X and Y offset relative */
            WORD Y;                 /* to the image's IA_Left  */
        } imp_Offset;               /* and IA_Top attributes.  */
    };
```

The imageclass dispatcher calls the graphics.library function EraseRect() to erase the image. The imageclass dispatcher gets the position of the image using the offsets from the IM_ERASE message and the dimensions it finds in the object's Image structure. The imageclass dispatcher does not do any bounds checking before calling EraseRect().

1.29 B / imageclass / New Methods: IM_DRAWFRAME

The IM_DRAWFRAME method instructs an image object to render itself within the confines of a given rectangle. The return value for this method is not explicitly defined.

This method uses a custom message structure that is basically an extension of the IM_DRAW message: impDraw

```
        {
        ULONG           MethodID;    /* IM_DRAWFRAME */
        struct RastPort *imp_RPort;  /* RastPort to render into */
        struct
        {
            WORD X;                  /* X and Y offset relative to the */
            WORD Y;           /* image's IA_Left and IA_Top attributes */
        } imp_Offset;
        ULONG    imp_State;  /* Visual state of image (see defines below)*/
        struct DrawInfo *imp_DrInfo;
                    /* DrawInfo describing target RastPort (can be NULL) */
        struct
        {
            WORD Width;   /* scale, clip, restrict, etc. to these bounds */
            WORD Height;
        } imp_Dimensions;
    };
```

The Width and Height fields provide the object's rectangular bounds. How the image object deals with the frame is implementation specific. If the imageclass dispatcher sees this message, it will convert it to an IM_DRAW message and send it back to the image's true class. An image subclass which assigns proper meaning to this method (i.e., frameiclass) should

handle this method itself.

This method is useful to classes of images that can scale or clip themselves to arbitrary dimensions. Typically, an instance of a class that truly supports this method will massage its imagery as best it can to fit into the rectangle.

In general, applications that use this method to draw an object should use the
IM_ERASEFRAME
method to erase it (see below). This will ensure that the image erases itself at the proper scale.

1.30 B / imageclass / New Methods: IM_HITFRAME

This method is a special version of
IM_HITTEST
for images that support

IM_DRAWFRAME
. It asks an image if a point would be inside it if the image was confined (scaled, clipped, etc.) to a rectangular bounds. The return value for this method is not explicitly defined.

This method uses a custom message structure:

```
struct impHitTest
{
    ULONG MethodID;    /* IM_HITFRAME */
    struct
    {
        WORD X;        /* Coordinates of point to test for hit */
        WORD Y;
    } imp_Point;

    struct
    {
        WORD Width;  /* scale, clip, restrict, etc. to these bounds */
        WORD Height;
    } imp_Dimensions;
};
```

The
imageclass
dispatcher treats IM_HITFRAME just like
IM_HITTEST
,
ignoring the restricting dimensions.

1.31 B / imageclass / New Methods: IM_ERASEFRAME

This method is a special version of
IM_ERASE
for images that support

IM_DRAWFRAME
. It asks an image to erase itself as if it were confined (scaled, clipped, etc.) to a rectangular bounds. The return value for this method is not explicitly defined.

This method uses a custom message structure:

```
struct impErase /* NOTE: This is a subset of impDraw */
    {
        ULONG           MethodID;   /* IM_ERASEFRAME */
        struct RastPort *imp_RPort; /* The image's RastPort */
        struct
        {
            WORD X;      /* X and Y offset relative to the */
            WORD Y;      /* image's IA_Left and IA_Top attributes */
        } imp_Offset;

        struct
        {
            WORD Width;  /* scale, clip, restrict, etc. to these bounds */
            WORD Height;
        } imp_Dimensions;
    };
```

The
imageclass
dispatcher handles an IM_ERASEFRAME message as if it was an IM_ERASE message, ignoring the bounds. See the imageclass description for

IM_ERASE
for more details.

1.32 B / imageclass / New Methods: IM_FRAMEBOX

This method applies to image classes that are used to put a frame ↩
centered
around some other objects. This method asks a framing image what its dimensions should be if it had to frame some object or set of objects that fit into a rectangular bounds. For example, to draw an
frameiclass
image
around a group of gadgets that fit into a specific rectangle, you first send the frameiclass object an IM_FRAMEBOX message describing the dimensions and position of that rectangle. The frame reports what its position and dimensions would have to be to surround those gadgets. Use these results to draw the frameiclass image. The return value for this method is not explicitly defined.

IM_FRAMEBOX uses a custom message structure:

```
    struct impFrameBox
    {
        ULONG           MethodID;         /* IM_FRAMEBOX  */
        struct IBox     *imp_ContentsBox; /* The object fills in this  */
                                          /* structure with the        */
                                          /* dimensions of a rectangle */
                                          /* big enough to frame...    */
        struct IBox     *imp_FrameBox;    /* <------- this rectangle.  */
        struct DrawInfo *imp_DrInfo;      /* imp_DrInfo can be NULL.   */
        ULONG           imp_FrameFlags;
    };
    #define FRAMEF_SPECIFY  (1<<0)  /* Make do with the dimensions */
                                    /* passed in FrameBox.         */
```

The imp_FrameBox field points to an IBox structure (defined in <intuition/intuition.h>) describing the dimensions and position of the rectangle to frame. After the framing image determines the position and size it should be in order to properly frame imp_FrameBox, it stores the result in the imp_ContentsBox IBox. This method allows an application to use a framing image without having to worry about image specific details such as the thickness of the frame or centering the frame around the object.

The imp_FrameFlags field is a bit field used to specify certain options for the IM_FRAMEBOX method. Currently, there is only one defined for it, FRAMEF_SPECIFY. If this bit is set, the imp_FrameBox contains a width and height that the frame image has to use as its width and height, even if the imp_FrameBox is smaller than imp_ContentsBox. The frame is free to adjust its position, but it is stuck with the imp_FrameBox dimensions. This allows an application to set the size of the frame image and still allow the frame image to position itself so it is centered on a rectangle.

The
imageclass
dispatcher does not support this method. It returns zero.

1.33 B / imageclass / Changed Methods: OM_NEW

The instance data for
imageclass
contains an Image structure, and its Depth field is initialized to CUSTOMIMAGEDEPTH, which identifies such images to Intuition. The Image's Width and Height fields default to arbitrary positive numbers for safety, but an imageclass subclass or an application should set these attributes to something meaningful.

1.34 B / imageclass / Changed Methods: OM_SET

```
                This method applies to all
                imageclass
                 attributes.  OM_SET returns 1.
```

1.35 B / imageclass / Attributes:IA_Left, IA_Top, IA_Width, IA_Height (ISG)

```
                These attributes correspond to similarly named fields in the  ↩
                   Intuition
Image structure.  The
                imageclass
                 dispatcher stores these attributes in
their corresponding fields in the image object's embedded Image structure.
```

1.36 B / imageclass / Attributes: IA_FGPen, IA_BGPen (ISG)

```
These attributes are copied into the Image structure's PlanePick and
PlaneOnOff fields, respectively.
```

1.37 B / imageclass / Attributes: IA_Data (ISG)

```
A pointer to general image data.  This value is stored in the ImageData
field of the Image structure.
```

1.38 B / imageclass / Attributes: IA_Pens ()

```
                This attribute points to an alternative pen array for the image.

                Imageclass
                 does not support this attribute, it is described here for
subclasses to use.  See the "Intuition Screens" chapter of the Amiga ROM
Kernel Reference Manual: Libraries for more information on the pen array.
```

1.39 B Boopsi Class Reference / frameiclass

```
                Class:                  frameiclass
Superclass:
                imageclass
                Include File:           <intuition/imageclass.h>

This is a class of framing image, which can optionally fill itself.  Its
purpose is to frame other display elements using an embossed or recessed
```

```
rectangular frame.  The frame renders itself using the appropriate
DrawInfo pens (SHINEPEN, SHADOWPEN, etc.).  This class is intelligent
enough to bound or center its contents.


New Methods:
------------
None.

Changed Methods:
----------------

                 IM_DRAW

                 IDS_NORMAL, IDS_INACTIVENORMAL, IDS_DISABLED

                 IDS_SELECTED, IDS_INACTIVESELECTED

                 IM_DRAWFRAME

                 IM_FRAMEBOX
                Attributes:
-----------

                 IA_Recessed (IS)

                 IA_EdgesOnly (IS)
```

1.40 B / frameiclass / Changed Methods: IM_DRAW

```
                This method tells a
                frameiclass
                 object to render itself using the position
and dimensions of its Image structure.  It supports two sets of drawing
states (passes in the
                impDraw.imp_State
                 field):
```

1.41 B / frameiclass / IDS_NORMAL, IDS_INACTIVENORMAL, IDS_DISABLED

```
In these states, the frame renders its edges using SHADOWPEN and SHINEPEN.
If it is a filled frame the frame uses the BACKGROUNDPEN for its interior.
Note that the frame renders the same imagery for all three of these states.
```

1.42 B / frameiclass / Changed Methods: IDS_SELECTED, IDS_INACTIVESELECTED

```
                In these states, the frame renders its edges using SHADOWPEN and ↩
                   SHINEPEN.
If it is a filled frame the frame uses the FILLPEN for its interior.

See the
                imageclass
                 description for
                IM_DRAW
                 for more details.
```

1.43 B / frameiclass / Changed Methods: IM_DRAWFRAME

```
                This method is almost the same as the
                frameiclass

                IM_DRAW
                 method, except
this method accepts a width and height that overrides the width and height
stored in the object's Image structure.  It uses the same drawing states
as the frameiclass's IM_DRAW method.  See the
                imageclass
                 description for

                IM_DRAWFRAME
                 for more information.
```

1.44 B / frameiclass / Changed Methods: IM_FRAMEBOX

```
                This method asks a
                frameiclass
                 image what its dimensions would be if it
has to frame a specific rectangular area.  See the
                imageclass
                 description
for
                IM_FRAMEBOX
                 for more information.
```

1.45 B / frameiclass / Attributes: IA_Recessed (IS)

```
                If this attribute is TRUE, a
                frameiclass
                 object will appear recessed into
the drawing surface.  It does this by swapping its use of the SHADOWPEN
and SHINEPEN.  By default, the frame appears to be raised from the surface.
```

1.46 B / frameiclass / Attributes: IA_EdgesOnly (IS)

```
If this attribute is TRUE, the frame does not fill itself, it just draws
its edges.
```

1.47 B Boopsi Class Reference / sysiclass

```
                Class:                  sysiclass
Superclass:
                imageclass
                Include File:           <intuition/imageclass.h>

This is a class of system images and standard application images.  As of
Intuition version 37, there are 11 possible sysiclass image glyphs to
choose from:

    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.
    RIGHTIMAGE   Right arrow image.
    UPIMAGE      Up arrow image.
    DOWNIMAGE    Down arrow image.
    CHECKIMAGE   Checkmark image.
    MXIMAGE      Radio button image.

The class caches the image's bitmap to improve rendering speed.

New Methods:
------------
None.

Changed Methods:
----------------
None.


Attributes:
-----------

                SYSIA_DrawInfo (I)

                SYSIA_Which (I)

                SYSIA_Size (I)
```

1.48 B / sysiclass / Attributes: SYSIA_DrawInfo (I)

This attribute contains a pointer to a DrawInfo structure (defined in
<intuition/screens.h>) describing the target screen. The class requires
this attribute in order to generate the image into a bitmap cache.

1.49 B / sysiclass / Attributes: SYSIA_Which (I)

This attribute identifies which of the system image glyphs the
sysiclass
object uses. It can be one of the 11 glyphs
described above
.

1.50 B / sysiclass / Attributes: SYSIA_Size (I)

This attribute identifies which image size to use for the object. ↩
This
generalizes Intuition's older concept of two different system image
dimensions. There are three possible values for this attribute:

```
    SYSISIZE_MEDRES  Meant for Hires, non-interlaced 640x200/256 display.
    SYSISIZE_HIRES   Meant for Hires, interlaced 640x400/512 display.
    SYSISIZE_LOWRES  Meant for Lores 320x200/256 display.
```

These sizes do not apply to all of the glyphs consistently. See the chart
below for image dimensions (width x height) according to the SYSIA_Size
and the glyph type. An 'H' for the height means the glyph allows its
height to be specified with
IA_Height
.

SYSISIZE	LOWRES	SYSISIZE_MEDRES	SYSISIZE_HIRES
DEPTHIMAGE	18 x H	24 x H	24 x H
ZOOMIMAGE	18 x H	24 x H	24 x H
SIZEIMAGE	13 x 11	18 x 10	18 x 10
CLOSEIMAGE	15 x H	20 x H	20 x H
SDEPTHIMAGE	17 x H	23 x H	23 x H
LEFTIMAGE	16 x 11	16 x 10	23 x 22
RIGHTIMAGE	16 x 11	16 x 10	23 x 22
UPIMAGE	13 x 11	18 x 11	23 x 22
DOWNIMAGE	13 x 11	18 x 11	23 x 22
CHECKIMAGE	26 x 11	26 x 11	26 x 11
MXIMAGE	17 x 9	17 x 9	17 x 9

1.51 B Boopsi Class Reference / fillrectclass

```
                Class:                   fillrectclass
Superclass:
                imageclass
                Include File:            <intuition/imageclass.h>

This is a class of filled rectangles.  The fillrectclass object can use a
pattern to fill in its interior.

New Methods:
------------
None.

Changed Methods:
----------------

                 IM_DRAW

                 IM_DRAW
                Attributes:
-----------

                 IA_APattern, IA_APatSize (IS)

                 IA_Mode (IS)
```

1.52 B / fillrectclass / Changed Methods: IM_DRAW

```
                This method asks a
                fillrectclass
                 object to render itself relative to the
position (LeftEdge and TopEdge) and dimensions (Width and Height) in its
embedded Image structure.  See the
                imageclass
                 description of
                IM_DRAW
                 for
more details.
```

1.53 B / fillrectclass / Changed Methods: IM_DRAWFRAME

```
                This method asks a
                fillrectclass
                 object to render itself relative to the
position in its embedded Image structure, but using the width and height
passed in the message's Dimensions.Width and Dimensions.Height fields.
See the
                imageclass
                 description of
```

IM_DRAWFRAME
 for more details.

1.54 B / fillrectclass / Attributes: IA_APattern, IA_APatSize (IS)

These attributes supply the
fillrectclass
 object with an area fill
pattern. The IA_APattern attribute points to the area fill pattern for the object. The IA_APatSize attribute is the depth of the area fill pattern. These attribute values are similar to the parameters passed to the SetAfPt() macro (defined in <graphics/gfxmacros.h>) and indirectly correspond to fields in a RastPort structure. For more information on these patterns, see the section on patterns in the "Graphics Primitives" chapter of the Amiga ROM Kernel Reference Manual: Libraries.

1.55 B / fillrectclass / Attributes: IA_Mode (IS)

This attribute contains the drawing mode for the pattern (JAM1, JAM2, etc.)

1.56 B Boopsi Class Reference / itexticlass

Class: itexticlass
Superclass:
imageclass
Include File: <intuition/imageclass.h>

This is a class of image objects that render an IntuiText structure. Using some of the
imageclass
 attributes, the object can override some of
the parameters in the IntuiText structure. This class makes it easy to share an IntuiText structure between objects.

New Methods:

None.

Changed Methods:

IM_DRAW/IM_DRAWFRAME

1.57 B / itexticlass / New Methods: IM_DRAW/IM_DRAWFRAME

```
                These methods ask an
                itexticlass
                 object to render its IntuiText structure,
which it gets from the
                imageclass

                IA_Data
                 attribute.  An itexticlass
object renders its IntuiText relative to the
                IA_Left
                 and
                IA_Top
                 attributes
it inherits from imageclass.  This method uses the JAM1 drawing mode and
the
                IA_FGPen
                 to render the text.  See the imageclass description of

                IM_DRAW
                /
                IM_DRAWFRAME
                 for more details.
```

1.58 B Boopsi Class Reference / gadgetclass

```
                Class:                  gadgetclass
Superclass:
                rootclass
                Include File:           <intuition/gadgetclass.h>

This is a base class for Intuition compatible gadget objects.  The
dispatcher for this class takes care of creating an Intuition Gadget
structure as part of its local instance data.  All of the standard Boopsi
gadget classes build on this class.  Normally there are no direct
instances of this class, only instances of subclasses of gadgetclass.

The behavior of a Boopsi gadget depends on how it handles the five Boopsi
gadget methods: GM_HITTEST, GM_RENDER, GM_GOACTIVE, GM_HANDLEINPUT, and
GM_GOINACTIVE.  Intuition controls a Boopsi gadget by sending it these
types of messages.  The structures that these methods use for their
messages begin with the method's ID followed by a pointer to a GadgetInfo
structure (defined in <intuition/cghooks.h>).  The GadgetInfo structure is
a read-only structure that contains information about the gadget's
rendering environment.  The gadget uses this to find things like its
window, screen, or pen array.  Although this structure does contain a
pointer to a RastPort for the gadget, the gadget must not use this
RastPort for rendering.  The gadget can obtain a RastPort for rendering by
calling the Intuition function ObtainGIRPort() using the GadgetInfo
structure.  See the intuition.library Autodocs for more details on this
function.
```

These methods are not defined directly by gadgetclass. It is up to subclasses of gadgetclass to implement them.

Like all Boopsi methods, these methods run on the context of the task that called the method. Normally, Intuition is the only entity that calls these methods, so these normally operate in the input.device's task. Because a gadget may have to process a large number of input events, poor implementations of gadget methods (especially the GM_HANDLEINPUT method) can degrade system performance.

New Methods:

GM_HITTEST

GM_GOACTIVE

GM_GOINACTIVE

GM_RENDER

GM_HANDLEINPUT

Changed Methods:

OM_NEW

OM_NOTIFY

Attributes:

GA_Previous (I)

ICA_TARGET (IS)

ICA_MAP (IS)

GA_Left, GA_Top, GA_Width, GA_Height (IS)

GA_RelRight, GA_RelBottom, GA_RelWidth, GA_RelHeight (IS)

The remaining attributes defined by gadgetclass are used to set ↩ the fields in the Gadget structure of the Boopsi gadget. Some Boopsi gadgets do not pay attention to many of the fields in its Gadget structure, so most applications will not have to worry about the majority of these attributes. Some gadget classes assign special meanings to these attributes. See the documentation of the specific gadget classes for more details.

GA_IntuiText, GA_Text, GA_LabelImage (IS)

GA_Image (IS)

GA_Border, GA_SelectRender, GA_ID, GA_UserData, GA_SpecialInfo (↩ IS)

```
GA_GZZGadget, GA_SysGadget (IS)

GA_Disabled, GA_Selected (IS)

GA_EndGadget, GA_Immediate, GA_RelVerify, GA_FollowMouse, (IS)

GA_RightBorder, GA_LeftBorder, GA_TopBorder, GA_BottomBorder, (IS ↩
    )

GA_ToggleSelect, GA_TabCycle (IS)

GA_Highlight (IS)

GA_SysGType (IS)
```

1.59 B / gadgetclass / New Methods: GM_HITTEST

This method asks a gadget if a point is within its bounds. Usually the point corresponds to a mouse click. Intuition sends a gadget this message when the user clicks inside the rectangular bounds found in the object's Gadget structure (using its TopEdge, LeftEdge, Width, and Height fields). This method returns GMR_GADGETHIT if a point is within the gadget, otherwise it returns zero. Because the gadget decides if it was hit, the gadget can be almost any shape or pattern. Boopsi gadgets that default to using the bounds of their Gadget structure should always return GMR_GADGETHIT.

GM_HITTEST uses a custom message structure (defined in <intuition/gadgdetclass.h>):

```
    struct gpHitTest
    {
        ULONG             MethodID;     /* GM_HITTEST   */
        struct GadgetInfo *gpht_GInfo;
        struct
        {
            WORD X;                     /* Is this point inside */
            WORD Y;                     /* of the gadget?       */
        } gpht_Mouse;
    };
```

The gpht_Mouse.X and gpht_Mouse.Y fields make up the X and Y coordinates of the hit point. These coordinates are relative to the upper-left corner of the gadget (Gadget.LeftEdge, Gadget.TopEdge).

1.60 B / gadgetclass / New Methods: GM_RENDER

This method tells a gadget to render itself. The return value for this method is not explicitly defined.

GM_RENDER uses a custom message structure (defined in

<intuition/gadgetclass.h>):

```
    struct gpRender
    {
        ULONG             MethodID;   /* GM_RENDER */
        struct GadgetInfo *gpr_GInfo;
        struct RastPort   *gpr_RPort; /* all ready for use */
        LONG              gpr_Redraw; /* might be a "highlight pass" */
    };
```

The GM_RENDER message contains a pointer to the Gadget's RastPort which it can use for rendering. The Gadget renders itself according to how much imagery it needs to replace. The gpr_Redraw field contains one of three values:

GREDRAW_REDRAW Redraw the entire gadget.

GREDRAW_UPDATE The user has manipulated the gadget changing the imagery. Update only that part of the gadget's imagery that is effected by the user manipulating the gadget (for example, the knob and scrolling field of the prop gadget).

GREDRAW_TOGGLE If this gadget supports it, toggle to or from the highlighting imagery.

1.61 B / gadgetclass / New Methods: GM_GOACTIVE

This method asks a gadget if it is OK to make it the active gadget ↩
. The
active gadget is the gadget that is currently receiving user input. Intuition sends this message after a gadget responds affirmatively to the

GM_HITTEST
method. A gadget becomes active because it needs to process input events (like a prop gadget or string gadget).

Some types of gadget do not need to become active. These gadgets do not have to process input from the user, they only have to deal with a single mouse click to toggle their state. Because that mouse click triggered this method, the button already has all of the user input it requires. Note that the behavior of the GadTools button differs from a Boopsi buttongclass gadget, which processes other input events besides a single mouse click. See the entry for
buttongclass
in this Appendix for more details.

GM_GOACTIVE uses a custom message structure (defined in <intuition/gadgetclass.h>):

```
    struct gpInput
    {
        ULONG             MethodID;    /* GM_GOACTIVE or GM_HANDLEINPUT */
        struct GadgetInfo *gpi_GInfo;
```

```
        struct InputEvent *gpi_IEvent; /* The input event that triggered */
                                       /* this method (can be NULL for   */
                                       /* GM_GOACTIVE).                  */
        LONG         *gpi_Termination; /* For GADGETUP IntuiMessage.Code */
        struct
        {
            WORD X;             /* Mouse position relative to upper      */
            WORD Y;             /* left corner of gadget (LeftEdge,      */
        } gpi_Mouse;            /* TopEdge).                             */
    };
```

The gpi_IEvent field points to the struct InputEvent that triggered the GM_GOACTIVE message. If gpi_IEvent is NULL, the GM_GOACTIVE message was triggered by a function like intuition.library's ActivateGadget() and not by the user clicking the gadget.

For gadgets that only want to become active as a direct result of a mouse click, this difference is important. For example, the prop gadget becomes active only when the user clicks on its knob. Because the only way the user can control the prop gadget is via the mouse, it would not make sense for it to be activated by anything besides the mouse. On the other hand, a string gadget gets input from the keyboard, so a string gadget doesn't care what activates it. Its input comes from the keyboard rather than the mouse.

A gadget's GM_GOACTIVE method returns GMR_MEACTIVE (defined in <intuition/gadgetclass.h>) if it wants to be the active gadget. Otherwise it returns
GMR_NOREUSE
. For a description of what these values mean, See their description in the
gadgetclass
's
GM_HANDLEINPUT
method, below.

If necessary, a gadget's GM_GOACTIVE method can precalculate and cache information before it becomes the active gadget. The gadget will use this information while it's processing user input with the
GM_HANDLEINPUT
method. When it is time for the active gadget to become inactive, Intuition will send the gadget a GM_GOINACTIVE message. The gadget can clean up its precalculations and cache in the GM_GOINACTIVE method. For more information on GM_GOINACTIVE, see its description below.

1.62 B / gadgetclass / New Methods: GM_HANDLEINPUT

This method asks an active gadget to handle an input event. After Intuition gets an OK to make this gadget object active (see the

GM_GOACTIVE
method above), Intuition starts sending input events to the gadget. Intuition sends them in the form of a GM_HANDLEINPUT message. This method uses the same custom message structure as GM_GOACTIVE (see the

```
                gpInput
                 structure above).

The information in the
                gpInput
                 structure is the same for GM_HANDLEINPUT as
it is for
                GM_GOACTIVE
                .  The only difference is that the GM_HANDLEINPUT
message's
                gpi_IEvent
                 can never be NULL.  It always points to an InputEvent
structure.

The gadget has to examine the incoming InputEvents to see how its state
may have changed.  For example,  a string gadget processes key presses,
inserting them into the gadgets string.  When the string changes, the
gadget has to update its visual state to reflect that change.  Another
example is the prop gadget.  If the user picks up the prop gadget's knob,
the prop gadget has to track the mouse to process changes to the gadget's
internal values.  It does this by processing IECLASS_RAWMOUSE events.

If the GM_HANDLEINPUT method needs to do some rendering, it must call
ObtainGIRPort() on the GM_HANDLEINPUT message's
                gpi_GInfo
                 to get a pointer
to a RastPort.  To relinquish this RastPort, the GM_HANDLEINPUT method
must call ReleaseGIRPort().  The GM_HANDLEINPUT method has to allocate and
release this RastPort, it cannot be cached in the
                GM_GOACTIVE
                 method.

The return value from GM_HANDLEINPUT informs Intuition if the gadget wants
to remain active.  The return values for the GM_HANDLEINPUT are similar to

                GM_GOACTIVE
                .  The gadget tells Intuition that it wants to remain active by
returning GMR_MEACTIVE.  A gadget tells Intuition it wants to become
inactive by returning one of the "go inactive" return values:

GMR_NOREUSE     Tells Intuition to throw away the
                gpInput.gpi_IEvent
                                InputEvent.

GMR_REUSE       Tells Intuition to reprocess the
                gpInput.gpi_IEvent
                                InputEvent after deactivating the gadget.

GMR_NEXTACTIVE  Tells Intuition to throw away the
                gpInput.gpi_IEvent
                                InputEvent and activate the next GFLG_TABCYCLE  ↩
                                   gadget.

GMR_PREVACTIVE  Tells Intuition to throw away the
                gpInput.gpi_IEvent
```

InputEvent and activate the previous GFLG_TABCYCLE ↩
gadget.

GMR_NOREUSE tells Intuition that the gadget does not want to be active and should throw away the InputEvent that triggered the GM_HANDLEINPUT message (or the

GM_GOACTIVE
message). For example, an active prop gadget returns GMR_NOREUSE when the user lets go of the left mouse button (thus letting go of the prop gadget's knob).

A gadget can also return GMR_REUSE, which tells Intuition to reuse the InputEvent. For example, if the user clicks outside of an active string gadget, that string gadget returns GMR_REUSE so Intuition can process that mouse click, which could be over another gadget. Another case where a string gadget returns GMR_REUSE is when the user pushes the right mouse button (the menu button). The string gadget becomes inactive and the menu button InputEvent gets reused by Intuition so it can pop up the menu bar.

The other two possible return values, GMR_NEXTACTIVE and GMR_PREVACTIVE were added to the OS for Release 2.04. These tell Intuition that a gadget no longer wants to be active and that the GM_HANDLEINPUT message InputEvent should be discarded. Intuition then looks for the next non-disabled (GMR_NEXTACTIVE) or previous (GMR_PREVACTIVE) gadget that has its GFLG_TABCYCLE flag set in its Gadget.Activation field (see the

gadgetclass

GA_TabCycle
attribute below), and attempts to activate it.

For both

GM_GOACTIVE
and GM_HANDLEINPUT, the gadget can bitwise OR any of these "go inactive" return values with GMR_VERIFY. The GMR_VERIFY flag tells Intuition to send a IDCMP_GADGETUP IntuiMessage to the gadget's window. If the gadget uses GMR_VERIFY, it has to supply a value for the IntuiMessage's Code field. It does this by passing a value in the

gpInput
's gpi_Termination field. This field points to a long word, the lower 16-bits of which Intuition copies into the Code field. The upper 16-bits are for future enhancements, so clear these bits.

1.63 B / gadgetclass / New Methods: GM_GOINACTIVE

This method tells the active gadget to become inactive. The ↩
return value
for this method is not explicitly defined.

GM_GOINACTIVE uses a custom message structure (defined in <intuition/gadgetclass.h>):

```
struct gpGoInactive
```

```
    {
        ULONG             MethodID;    /* GM_GOINACTIVE */
        struct GadgetInfo *gpgi_GInfo;

    /* V37 field only!  DO NOT attempt to read under V36! */
        ULONG             gpgi_Abort; /* gpgi_Abort=1 if gadget was    */
                                      /* aborted by Intuition and 0 if */
                                      /* gadget went inactive at its   */
                                      /* own request.                  */
    };
```

The gpgi_Abort field contains either a 0 or 1. If it is 0, the gadget became inactive at its own request (because the GM_HANDLEINPUT method returned something besides GMR_MEACTIVE). If gpgi_Abort is 1, Intuition aborted this active gadget. Some cases where Intuition aborts a gadget include: the user clicked in another window or screen, an application removed the active gadget with RemoveGList(), and an application called ActivateWindow() on a window other than the gadget's window.

If the gadget allocated any resources to cache or precalculate information in the GM_GOACTIVE method, it should deallocate those resources in this method.

1.64 B / gadgetclass / Changed Methods: OM_NEW

This method allocates space for an embedded struct Gadget (defined in <intuition/intuition.h>) and initializes some of the attributes defined by gadgetclass.

1.65 B / gadgetclass / Changed Methods: OM_NOTIFY

This method tells a gadget to send an OM_UPDATE message to its target object. Boopsi gadgets have a function similar to icclass objects--each gadget can have an ICA_TARGET and ICA_MAP

```
                in order to notify some target
object of attribute changes.  When a Boopsi gadget sends an OM_NOTIFY
message, it always includes its GA_ID.  This makes it easy for an
application to tell which gadget initially sent the OM_NOTIFY.  See the
description of icclass's
                OM_NOTIFY
                 and
                OM_UPDATE
                 and the
                rootclass
                's

                OM_NOTIFY
                 and
                OM_UPDATE
                 methods for more details.
```

1.66 B / gadgetclass / Attributes: GA_Previous (I)

```
                This attribute is used to insert a new gadget into a list of  ↩
                   gadgets
linked by their Gadget.NextGadget field.  When the
                OM_NEW
                 method creates
the new gadget, it inserts the new gadget into the list following the
GA_Previous gadget.  This attribute is a pointer to the gadget  (struct
Gadget *) that the new gadget will follow.  This attribute cannot be used
to link new gadgets into the gadget list of an open window or requester,
use AddGList() instead.
```

1.67 B / gadgetclass / Attributes: ICA_TARGET (IS)

```
                This attribute stores the address of the gadget's target object.   ↩
                   Whenever
the gadget receives an
                OM_NOTIFY
                 message, it sends an
                OM_UPDATE
                 message to
its target.  If the gadget has an attribute mapping list (see the
                ICA_MAP
                attribute below), it also maps the IDs from the OM_NOTIFY message.

If the value of ICA_TARGET is ICTARGET_IDCMP, the gadget sends an
IDCMP_IDCMPUPDATE IntuiMessage to its window.  See the
                rootclass
                description of
                OM_UPDATE
                 for more information.
```

1.68 B / gadgetclass / Attributes: ICA_MAP (IS)

```
                This attribute points to a tag list of attribute mappings which ↩
                   the gadget
uses to change the attribute IDs of an
                OM_UPDATE
                's attribute/value pairs.
For example, if a gadget had the following ICA_MAP:

    struct TagItem map[] =
    {
        {PGA_Top, STRINGA_LongVal},
        {MYATTR, MYNEWATTR},
        {TAG_END, }
    };

before it sends an
                OM_UPDATE
                 to its
                ICA_TARGET
                , the gadget scans through
the OM_UPDATE message's attribute/value pairs looking for the
                PGA_Top
                 and
MYATTR attributes.  If it finds the PGA_Top attribute, it changes PGA_Top
to
                STRINGA_LongVal
                .  Likewise, if the gadget finds the MYATTR attribute,
it changes MYATTR to MYNEWATTR.  The gadget does not disturb the
attribute's value, only its ID.
```

1.69 B / gadgetclass / Attributes:GA_Left, GA_Top, GA_Width, GA_Height (IS)

```
                These attributes correspond to the Gadget structure's LeftEdge, ↩
                   TopEdge,
Width, and Height fields.  Setting these clears the "
                gadget relative
                "
flags (below).
```

1.70 B / / GA_RelRight, GA_RelBottom, GA_RelWidth, GA_RelHeight (IS)

```
These attributes correspond to the Gadget structure's LeftEdge, TopEdge,
Width, and Height fields.  Setting any of these attributes also sets the
corresponding "relative" flag in the Gadget structure's Flags field
(respectively, GFLG_RELRIGHT, GFLG_RELBOTTOM, GFLG_RELWIDTH, and
GFLG_RELHEIGHT).  Note that the value passed in this attribute is normally
a negative LONG.  See the "Intuition Gadgets" chapter of the Amiga ROM
Kernel Reference Manual: Libraries for more information on these flags.
```

1.71 B / gadgetclass / Attributes:GA_IntuiText, GA_Text, GA_LabelImage (IS)

These attributes correspond to one field in the object's embedded Gadget structure--the GadgetText field. Setting any of these attributes copies the attribute's value blindly into the GadgetText field. In addition, setting GA_Text also sets the GFLG_LABELSTRING flag in Gadget.Flags and setting GA_LabelImage sets the GFLG_LABELIMAGE flag in Gadget.Flags. The GA_IntuiText attribute must 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. Note that most gadget classes do not support GA_Text and GA_LabelImage. See the description of specific gadget classes for more details.

1.72 B / gadgetclass / Attributes: GA_Image (IS)

This attibute is a pointer to either a Boopsi image or a Release 1.3-compatible Intuition image. This attribute corresponds to the Gadget's GadgetRender field. The gadgetclass dispatcher will not dispose of this image when it disposes of the gadget object.

1.73 B / gadgetclass / Attributes: GA_Border - GA_SpecialInfo (IS)

GA_Border, GA_SelectRender, GA_ID, GA_UserData, GA_SpecialInfo (IS) - These attributes correspond to the similarly named fields in the Gadget structure embedded in the gadget object.

1.74 B / gadgetclass / Attributes: GA_GZZGadget, GA_SysGadget (IS)

These are boolean attributes that correspond to the flags in the object's Gadget.GadgetType field. If the value passed with the attribute is TRUE, the corresponding flag in Gadget.GadgetType is set. If the value passed with the attribute is FALSE, the corresponding flag in Gadget.GadgetType is cleared. See the <intuition/intuition.h> include file or the "Intuition Gadgets" chapter of the Amiga ROM Kernel Reference Manual: Libraries for more information.

1.75 B / gadgetclass / Attributes: GA_Disabled, GA_Selected (IS)

These are boolean attributes that correspond to the similarly named flags in the object's Gadget.Flags field. If the value passed with the attribute is TRUE, the corresponding flag in Gadget.Flags is set. If the value passed with the attribute is FALSE, the corresponding flag in Gadget.Flags is cleared. See the <intuition/intuition.h> include file or the "Intuition Gadgets" chapter of the Amiga ROM Kernel Reference Manual: Libraries for more information.

1.76 B / gadgetclass / Attributes: GA_EndGadget - GA_TabCycle (IS)

```
GA_EndGadget, GA_Immediate, GA_RelVerify, GA_FollowMouse, (IS)
GA_RightBorder, GA_LeftBorder, GA_TopBorder, GA_BottomBorder, (IS)
GA_ToggleSelect, GA_TabCycle (IS) - These are boolean attributes that
correspond to the flags in the object's Gadget.Activation field.  If the
value passed with the attribute is TRUE, the corresponding flag in
Gadget.Activation is set.  If the value passed with the attribute is
FALSE, the corresponding flag in Gadget.Activation is cleared.  See the
<intuition/intuition.h> include file or the "Intuition Gadgets" chapter of
the Amiga ROM Kernel Reference Manual: Libraries for more information.
```

1.77 B / gadgetclass / Attributes: GA_Highlight (IS)

```
This attribute corresponds to the GFLG_GADGHIGHBITS portion of the
gadget's Gadget.Flags field.  This attribute can be one of four values
(from <intuition/intuition.h>):

    GFLG_GADGHCOMP, GFLG_GADGHBOX, GFLG_GADGHIMAGE, GFLG_GADGHNONE

See the "Intuition Gadgets" chapter of the Amiga ROM Kernel Reference
Manual: Libraries for more information.
```

1.78 B / gadgetclass / Attributes: GA_SysGType (IS)

```
This attribute corresponds to the system gadget type portion of the
gadget's Gadget.GadgetType fields.  This attribute is any one of the
following flags (from <intuition/intuition.h>):

    GTYP_SIZING, GTYP_WDRAGGING, GTYP_SDRAGGING, GTYP_WUPFRONT,
    GTYP_SUPFRONT, GTYP_WDOWNBACK, GTYP_SDOWNBACK, GTYP_CLOSE

See the "Intuition Gadgets" chapter of the Amiga ROM Kernel Reference
Manual: Libraries for more information.
```

1.79 B Boopsi Class Reference / propgclass

```
                Class:                  propgclass
Superclass:
                gadgetclass
                Include File:           <intuition/gadgetclass.h>

A Boopsi proportional ("prop") gadget.  The Boopsi prop gadget is similar
to the conventional prop gadget, but extends its function to make it
easier to use.  The Boopsi prop gadget keeps its current integer value in
its
                PGA_Top
                 attribute.
```

```
New Methods:
------------
None.

Changed Methods:
----------------

                 GM_HANDLEINPUT
               Attributes:
-----------

                 GA_Image (I)

                 PGA_Freedom (IG)

                 PGA_Top (ISGNU)

                 GA_Border (I)

                 PGA_NewLook (I)

                 PGA_Visible, PGA_Total (ISU)

                 GA_Highlight (I)

                 PGA_Borderless (I)
```

1.80 B / propgclass / Changed Methods: GM_HANDLEINPUT

```
               If the knob position changes sufficiently to change a
               propgclass
                object's

               PGA_Top
                attribute, the gadget will send an
               OM_NOTIFY
                message to itself,
which the propgclass dispatcher passes on to the
               gadgetclass
                dispatcher
for processing (see the
               rootclass
                description of
               OM_NOTIFY
                and
               OM_UPDATE
               for more information).

The
               OM_NOTIFY
                message will contain two attribute/value pairs,
               PGA_Top
                and
GA_ID.  While the prop gadget's PGA_Top is in a transitory state (while it
```

is active and the user is moving the prop gadget's knob), the gadget sends interim
OM_NOTIFY
messages. The interim OM_NOTIFY messages have the OPUF_INTERIM flag of the
opUpdate.opu_Flags
field set. When the user finishes manipulating the gadget (by letting go of the knob), the gadget sends a final OM_NOTIFY message, which has a cleared OPUF_INTERIM flag.

1.81 B / propgclass / Attributes: GA_Image (I)

Propgclass
intercepts this
gadgetclass
attribute before passing it on to gadgetclass. This attribute passes an image for the prop gadget's knob, which gets stored in the propgclass object's Gadget.Image structure. If the propgclass does not get a GA_Image when it creates a prop gadget, the prop gadget's knob defaults to an AUTOKNOB. An AUTOKNOB automatically sizes itself according to how large the range of the gadget is compared to the visible range of the gadget. See the
PGA_Visible
and
PGA_Total
attributes for more details.

1.82 B / propgclass / Attributes: GA_Border (I)

Propgclass
intercepts this
gadgetclass
attribute to prevent gadgetclass from setting up a border. If an application tries to set this attribute for a propgclass gadget, the prop gadget turns itself into an AUTOKNOB gadget.

1.83 B / propgclass / Attributes: GA_Highlight (I)

Propgclass
intercepts this
gadgetclass
attribute before passing it on to gadgetclass. It does this to make sure the highlighting is not set to GADGHBOX. GADGHBOX will be converted to GADGHCOMP. See the "Intuition Gadgets" chapter of the Amiga ROM Kernel Reference Manual: Libraries for more information on the types of gadget highlighting.

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

1.84 B / propgclass / Attributes: PGA_Freedom (IG)

```
                This attribute tells a
                propgclass
                 object on which axis the gadget's knob
is free to move, the horizontal or the vertical.  It is either FREEHORIZ
or FREEVERT.  The default is FREEVERT.
```

1.85 B / propgclass / Attributes: PGA_NewLook (I)

```
                This is a boolean attribute which corresponds to the PROPNEWLOOK  ↩
                   flag
PropInfo structure's Flags field (defined in <intuition/intuition.h>).  If
this attribute is TRUE, the new
                propgclass
                 object will use Release 2
imagery rather than the Release 1.3 imagery.
```

1.86 B / propgclass / Attributes: PGA_Borderless (I)

```
                This is a boolean attribute which corresponds to the  ↩
                   PROPBORDERLESS flag of
the PropInfo structure's Flags field (defined in <intuition/intuition.h>).
If this attribute is TRUE, the new
                propgclass
                 object will not have a
border around it.  In an AUTOKNOB propgclass gadget, if the PROPNEWLOOK
flag is set as well (see the
                PGA_NewLook
                 attribute), the knob will have a
3D appearance.
```

1.87 B / propgclass / PGA_Top (ISGNU), PGA_Visible, PGA_Total (ISU)

```
                These attributes replace the Pot and Body variables of the Release ↩
                    1.3
prop gadget. 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).

If the user clicks in the prop gadget but not on the knob, the entire knob
jumps one "page" (the size of the visible area minus one, PGA_Visible-1).
The page jump 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, sliding from 0 to PGA_Total - 1.

Note that when PGA_Top changes, the gadget sends itself an
                OM_NOTIFY
                message about this attribute change (see the
                propgclass
                 description of

                GM_HANDLEINPUT
                 for more information).  All three of these attributes have

                OM_UPDATE
                 access, so they can be controlled from other objects.
```

1.88 B Boopsi Class Reference / strgclass

```
                Class:                  strgclass
Superclass:
                gadgetclass
                Include File:           <intuition/gadgetclass.h>

Intuition compatible string gadgets.  The Boopsi string gadget can either
be a plain string gadget or an integer string gadget.  An integer gadget
filters out all characters except those that make up integer values.

New Methods:
------------
None.

Changed Methods:
----------------

                 OM_NEW
                Attributes:
-----------

                 STRINGA_LongVal (ISGNU)

                 STRINGA_TextVal (ISGNU)
                The remaining strgclass attributes correspond to the flags and ↩
                   fields that
the conventional Intuition string gadget uses.  See the "STRING GADGET
TYPE" section of the "Intuition Gadgets" chapter of the Amiga ROM Kernel
Reference Manual: Libraries for more information.


                 STRINGA_MaxChars, STRINGA_Buffer, (I)
```

```
STRINGA_UndoBuffer, STRINGA_WorkBuffer (I)

STRINGA_BufferPos, STRINGA_DispPos (ISU)

STRINGA_AltKeyMap (IS)

STRINGA_Font (IS)

STRINGA_Pens (IS)

STRINGA_ActivePens (IS)

STRINGA_EditHook (I)

STRINGA_EditModes (IS)

STRINGA_ReplaceMode, STRINGA_FixedFieldMode, STRINGA_NoFilterMode ↩
    (IS)

STRINGA_Justification (IS)

STRINGA_ExitHelp (IS)
```

1.89 B / strgclass / Changed Methods: OM_NEW

```
This method sets up the string gadget's StringInfo and  ↩
   StringExtend
structures.  It allocates 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: Foreground = 1,
Background = 0.  See the
rootclass
 description of the
OM_NEW
 method for
more details.
```

1.90 B / strgclass / Attributes: STRINGA_LongVal (ISGNU)

```
This attribute tells
strgclass
 that this gadget is an integer string
gadget and the new value of the integer is this attribute's value.
```

1.91 B / strgclass / Attributes: STRINGA_TextVal (ISGNU)

```
                This attribute tells
                strgclass
                 that this gadget is a plain string gadget.
The attribute points to a string which the object copies into the string
gadget's current string value buffer.

When a
                strgclass
                 gadget's internal
                STRINGA_LongVal
                 or STRINGA_TextVal
value changes (usually because the user manipulated the gadget), it sends
itself an
                OM_NOTIFY
                 message.  The OM_NOTIFY message will contain two
attribute/value pairs, GA_ID and either STRINGA_LongVal or STRINGA_TextVal
(depending on what kind of strgclass gadget it is).  Strgclass gadgets
only send a final OM_NOTIFY message (one's with the OPUF_INTERIM flag of
the
                opUpdate.opu_Flags
                 field cleared).
```

1.92 B / strgclass / Attributes: STRINGA_MaxChars - STRINGA_WorkBuffer (I)

```
STRINGA_MaxChars, STRINGA_Buffer, STRINGA_UndoBuffer, 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 (128 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.
```

1.93 B / strgclass / Attributes: STRINGA_BufferPos, STRINGA_DispPos (ISU)

```
The attributes tell the object its cursor and scroll position.
```

1.94 B / strgclass / Attributes: STRINGA_AltKeyMap (IS)

```
This attribute corresponds to the StringInfo.AltKeyMap field.
```

1.95 B / strgclass / Attributes: STRINGA_Font (IS)

```
This attributes points to an open TextFont structure that the string
gadget uses for rendering its text.
```

1.96 B / strgclass / Attributes: STRINGA_Pens (IS)

```
Pen numbers, packed as two WORDs into a longword, for rendering gadget
text.
```

1.97 B / strgclass / Attributes: STRINGA_ActivePens (IS)

```
Optional pen numbers, packed as two WORDs into a longword, for rendering
gadget text when the gadget is active.
```

1.98 B / strgclass / Attributes: STRINGA_EditHook (I)

```
Custom string gadget edit hook.
```

1.99 B / strgclass / Attributes: STRINGA_EditModes (IS)

```
Value taken from flags defined in <intuition/cghooks.h> for initial
editing modes.
```

1.100 B / strgclass / STRINGA_ReplaceMode - STRINGA_NoFilterMode (IS)

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

1.101 B / strgclass / Attributes: STRINGA_Justification (IS)

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

1.102 B / strgclass / Attributes: 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.
```

1.103 B Boopsi Class Reference / buttongclass

```
                Class:                  buttongclass
Superclass:
                gadgetclass
                Include File:           <intuition/gadgetclass.h>

A class of button gadget that continually sends interim
                OM_UPDATE
                 messages
to its target while the user holds down the button.  The button sends a
final OM_UPDATE message when the user lets  go of the button.  The imagery
for these objects is not built directly into the gadget.  Instead, a
buttongclass object uses a Boopsi image object, which it gets from its
GA_Image attribute.

New Methods:
------------
None.

Changed Methods:
----------------

                 GM_HITTEST

                 GM_HANDLEINPUT

                 GM_RENDER
                Attributes:
-----------

                 GA_IMAGE (IS)
```

1.104 B / buttongclass / Changed Methods: GM_HITTEST

```
                This method gets passed over to the button's image for processing. ↩
                     The
button's
                IM_HITTEST
                 checks for the hit.
```

1.105 B / buttongclass / Changed Methods: GM_HANDLEINPUT

```
                This method continuously issues
                OM_NOTIFY
                 messages for each IECLASS_TIMER
event it gets.  The OM_NOTIFY message's OPUF_INTERIM flag (from

                opUpdate.opu_Flags
                ) is set for all but the final OM_NOTIFY.
```

The
OM_NOTIFY
message contains one attribute/value pair, GA_ID. If the pointer is currently over the gadget image, the value of this attribute/value pair is the gadget's actual GA_ID (from the Gadget.GadgetID field). If the pointer isn't over the image, the value is the negative of the gadget's actual GA_ID.

1.106 B / buttongclass / Changed Methods: GM_RENDER

All rendering is passed along to the GadgetRender.Image (the GA_Image attribute). This method can tell its image to render in any of four image states:
IDS_INACTIVESELECTED, IDS_INACTIVENORMAL, IDS_SELECTED, or IDS_NORMAL.

1.107 B / buttongclass / Attributes: GA_Image (IS)

This attribute points to the gadget's Boopsi image. Changing this attribute will cause the gadget to refresh its imagery.

1.108 B Boopsi Class Reference / frbuttonclass

Class: frbuttonclass
Superclass: buttongclass
Include File: <intuition/gadgetclass.h>

This is a special class of button gadget that puts a Boopsi framing image around some other display element. This display element can be one of three things: plain text from the GA_Text attribute, an IntuiText from the GA_IntuiText attribute, or an Image from the GA_LabelImage attribute.

```
The user activates the gadget by clicking within the bounds of the
gadget's framing image, which it gets from the
                GA_Image
                 attribute.
Usually the framing image is an instance of an image class that supports
the IM_FRAMEBOX method (like
                frameiclass
                ).  If the framing image supports
the IM_FRAMEBOX method, the frbuttonclass object centers the frame image
around the display element.  See the
                imageclass
                 description of
                IM_FRAMEBOX
                for more information.

New Methods:
------------
None.

Changed Methods:
----------------

                 OM_NEW

                 GM_HITTEST

                 GM_RENDER
                Attributes:
-----------

                 GA_Width, GA_Height (S)

                 GA_DrawInfo (I)

                 GA_Text, GA_IntuiText, GA_LabelImage (IS)
```

1.109 B / frbuttonclass / Changed Methods: OM_NEW

```
                When this class creates an object, it sets the object's embedded
Gadget.Width and Gadget.Height fields according to the frame image in

                GA_Image
                .  If the GA_Image understands the
                IM_FRAMEBOX
                 method, the gadget
asks the GA_Image what it dimensions would be if it had to surround the
display element.  If the GA_Image does not support IM_FRAMEBOX, it just
copies the GA_Image image's width and height into the Gadget structure.
```

1.110 B / frbuttonclass / Changed Methods: GM_HITTEST

The gadget delegates this method to the framing image's IM_HITFRAME method.

1.111 B / frbuttonclass / Changed Methods: GM_RENDER

For this method, the frbuttonclass object first draws the framing image by sending the image an IM_DRAWFRAME message. The object then draws its display element.

1.112 B / frbuttonclass / Attributes: GA_Width, GA_Height (S)

These attribute correspond to the gadget's Width and Height fields ↩ . If the framing image supports IM_FRAMEBOX, changing these resizes the framing image. The framing image re-centers itself around the display element as best it can, and the frbuttonclass gadget re-renders the whole itself.

1.113 B / frbuttonclass / Attributes: GA_DrawInfo (I)

This attribute passes a pointer to a valid DrawInfo structure. If the

frbuttonclass gadget is going to frame plain text (passed to it in the

GA_Text attribute), the frbuttonclass gadget requires a DrawInfo ↩ structure to properly calculate the dimensions of the text.

1.114 B / frbuttonclass / Attributes:GA_Text,GA_IntuiText,GA_LabelImage (IS)

```
                These attributes tell the
                frbuttonclass
                 object what kind of imagery to use
as its display element.  See their description in the
                gadgetclass
                 entry
for more information.
```

1.115 B Boopsi Class Reference / groupgclass

```
                Class:                  groupgclass
Superclass:
                gadgetclass
                Include File:           <intuition/gadgetclass.h>

This is a class of objects that maintains an internal list of gadgets.
Its purpose is to make it easier to layout a group of gadgets.  Any
gadgets that are a member of a groupgclass object are rendered relative to
the groupgclass object's GA_Left and GA_Top attributes.  As new gadgets
are added to the groupgclass object, the groupgclass object's dimensions
grow to enclose the new gadgets.  When the groupgclass object receives an

                OM_DISPOSE
                 message, it not only disposes of itself, it also disposes of
all the gadgets in its list.  Groupgclass does not support the gadget
relative flags (
                GA_RelWidth, GA_RelHeight, GA_RelBottom, and GA_RelRight
                ).

New Methods:
------------
None.

Changed Methods:
----------------

                 OM_SET

                 GM_HITTEST

                 OM_ADDMEMBER

                 GM_RENDER

                 OM_REMMEMBER

                 GM_GOACTIVE/GM_GOINACTIVE/GM_HANDLEINPUT

                 OM_DISPOSE
                Attributes:
-----------
```

GA_Left, GA_Top (IS)

1.116 B / groupgclass / Changed Methods: OM_SET

This method 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.

1.117 B / groupgclass / Changed Methods: OM_ADDMEMBER

This method adds a gadget to the group object's list. The group object will increase the size of its select box to include the new gadget's select box. The group object moves the new member to an absolute location (by changing the new member's GA_Left and GA_Top attributes) relative to the group object's upper-left corner. Note that all members of the groupgclass object will be deleted by OM_DISPOSE.

1.118 B / groupgclass / Changed Methods: OM_REMMEMBER

This method removes a gadget added to the group object's list with OM_ADDMEMBER. Note that all members of the groupgclass object will be deleted by OM_DISPOSE.

1.119 B / groupgclass / Changed Methods: OM_DISPOSE

This method disposes of the groupgclass object and all its member gadgets.

1.120 B / groupgclass / Changed Methods: GM_HITTEST

This method works its way through the list of group members, sending each a GM_HITTEST message, looking for the first member in the list that says it has been hit. This member gadget becomes the active member.

1.121 B / groupgclass / Changed Methods: GM_RENDER

This method sends a GM_HITTEST message to each of its members.

1.122 B / groupgclass / Changed: GM_GOACTIVE/GM_GOINACTIVE/GM_HANDLEINPUT

This method passes the message to the active member's dispatcher for processing. For GM_GOINACTIVE and GM_HANDLEINPUT, the coordinates passed to the member's dispatcher in the message's gpi_Mouse.X and gpi_Mouse.Y fields are translated so that they are relative to the gadget's upper-left corner.

1.123 B / groupgclass / Attributes: GA_Left, GA_Top (IS)

These attributes correspond to the embedded Gadget.LeftEdge and Gadget.TopEdge fields. Setting these attributes in groupgclass object causes it to change its position as well as the position of each of the gadgets that have been added to the group gadget.