Boopsi in Release 3

by John Orr and Peter Cherna

For Release 3, Intuition gained significant improvements that directly affect Intuition's object oriented programming subsystem, Boopsi. Intuition now has gadget help features, bounding boxes for gadgets and their imagery, special custom layout gadgetry, and several other features that all have an impact on Boopsi.

This article assumes that the reader already has a basic understanding of Boopsi and the object oriented programming model. For those not comfortable with these concepts, see the ''Boopsi--Object Oriented Intuition'' chapter of the *ROM Kernel Reference Manual: Libraries*.

DoGadgetMethodA()

One Boopsi-related function added to *intuition.library* for Release 3 is DoGadgetMethodA():

```
ULONG DoGadgetMethodA(struct Gadget *object, struct Window *win,
                      struct Requester *req, Msg msg);
```

This function is similar to *amiga.lib*'s DoMethod(), except DoGadgetMethod() passes along the object's graphical environment in the form of a GadgetInfo structure. DoGadgetMethodA() gets this structure from the gadget's window (or requester). The GadgetInfo structure is important because a Boopsi gadget needs the information in that structure to render itself. Note that if you give DoGadgetMethodA() a `NULL` Window and `NULL` Requester pointer, DoGadgetMethodA() will pass a `NULL` GadgetInfo pointer.

The name and first parameter of DoGadgetMethodA() may lead you to believe that this function applies only to gadgets. Although you can certainly use this function on gadget objects, the function is not restricted to gadget objects. You can use this function on any Boopsi object. This is important for an object such as a model object, which may react to any Boopsi message by invoking some method on gadget objects connected to it. Because the model object receives environment information in the form of a GadgetInfo structure, the model can pass that information on to any objects connected to it.

Before this function, passing the environment information was not that easy. A good example of this is the *rkmmodelclass* example in the ‘‘Boopsi--Object Oriented Intuition’’ chapter of *RKRM: Libraries* (page 312-315). In that example, the rkmmodelclass is a subclass of modelclass. The rkmmodelclass object inherits the behavior of modelclass, so its sends updates to its member objects. One feature rkmmodelclass adds to modelclass is that these objects maintain an internal integer value. If that value changes, the rkmmodelclass object propagates that change to its member objects.

If one of the member objects happens to be a gadget object, changing the rkmmodelclass object’s internal value may change the visual state of the gadgets. For the gadget’s to update their visual state, they need the environment information from the GadgetInfo structure as well as the new internal value of the rkmmodelclass object. If an application used DoMethod() or SetAttrs() to set the rkmmodelclass object’s internal value, the rkmmodelclass object would not get a GadgetInfo structure. When the rkmmodelclass object propagates the internal value change to its member objects, it has no environment information to pass on to its member objects. As a result, the member gadgets can not update their visual state directly. This is particularly annoying for a propgclass object, because the visual state of the propgclass object can depend on the rkmmodelclass object’s integer value.

DoGadgetMethodA() corrects this problem. It passes a pointer to a GadgetInfo structure for the window (or requester) you pass to it. If you plan on implementing new Boopsi methods that need a GadgetInfo structure in their Boopsi message, make sure the second long word of the Boopsi message is a pointer to the GadgetInfo structure. DoGadgetMethodA() assumes that every method (except for `OM_NEW`, `OM_SET`, `OM_NOTIFY`, and `OM_UPDATE`; see the next paragraph) expects a GadgetInfo pointer in that place.

Iif you use DoGadgetMethodA() to send an `OM_SET` message to a Boopsi gadget, DoGadgetMethodA() passes a pointer to a GadgetInfo structure as the *third* long word in the Boopsi message. DoGadgetMethodA() is smart enough to know that the `OM_SET` method uses an opSet structure as its message, which has a pointer to a GadgetInfo structure as its *third* long word. DoGadgetMethodA() passes a GadgetInfo structure as the third parameter for the `OM_NEW`, `OM_SET`, `OM_NOTIFY`, and `OM_UPDATE` methods. For all other methods, like the gadgetclass methods, DoGadgetMethodA() passes a GadgetInfo structure as the *second* long word. For more information, see the V39 Autodoc for DoGadgetMethodA() and its varargs equivalent, DoGadgetMethod().

Bounding Boxes for Gadgets

Before Release 3, gadgets only had a select (or hit) box. This box is the area of the window that the user can click in to select the gadget. The select box limits certain gadgets because the gadget’s imagery could lie outside of the gadget’s hit box. For example, the hit box for a string gadget is the area that text appears when the user types into the gadget (see Figure 1). Many string gadgets have a border and a label, which the programmer sets using the Gadget structure’s GadgetRender and GadgetText fields. The border and label imagery appears outside of the hit box.

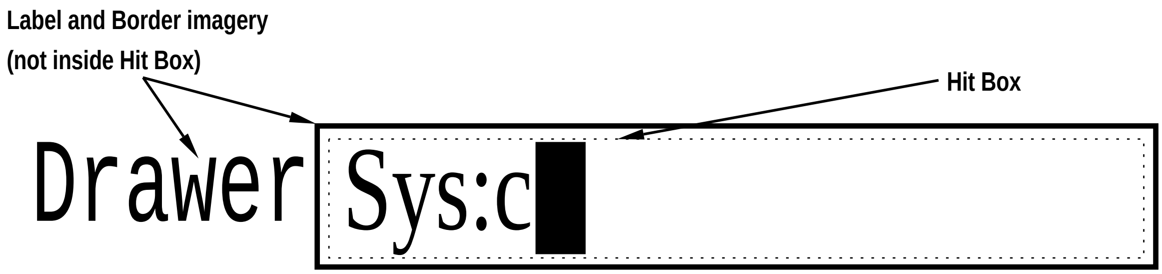

Figure 1 - String Gadget without Bounding Box

The major disadvantage of a gadget's imagery being external to its hit box has to do with relative gadgets (these are sometimes referred to as GREL gadgets). Intuition positions one of these gadgets relative to the edge's of the gadget's window. When the window changes dimensions, Intuition repositions the gadget within the window. The gadget can also make its size relative to the window's dimensions.

When Intuition resizes a window, it remembers which portions of the window's display sustained visual damage. When Intuition refreshes the visual state of the window, it only redisplays the parts of the window that sustained visual damage. When Intuition resizes a window that has a GREL gadget, Intuition erases the old gadget by erasing the hit box of the gadget and remembers that area as a region it will have to refresh. Intuition also figures out where the the new hit box will be and remembers that area as a region Intuition will have to refresh. Because Intuition will not erase or redraw any imagery that falls outside of the GREL gadget's hit box, Intuition will ignore any part of the gadget's label or border that is outside the hit box.

To remedy this situation, Intuition added a bounding box to gadgets. If a gadget has a bounding box, Intuition uses the bounding box in place of the hit box when figuring out what areas to refresh. With a bounding box, a gadget can extend its hit area to encompass all of its imagery.

The bounding box feature is not specific to Boopsi gadgets. Any Intuition gadget can have one. If the gadget doesn't supply a bounding box, Intuition will use the gadget's normal hit box as the bounding box, which yields the same result as previous versions of the OS.

Adding the bounding box feature to Intuition gadgets required extending the Gadget structure. There is a new structure called ExtGadget that is a superset of the old Gadget structure (from *<intuition/intuition.h>*):

```
struct ExtGadget
{
    /* The first fields match struct Gadget exactly */
    struct ExtGadget *NextGadget; /* Matches struct Gadget */
    WORD LeftEdge, TopEdge;       /* Matches struct Gadget */
    WORD Width, Height;           /* Matches struct Gadget */
    UWORD Flags;                  /* Matches struct Gadget */
    APTR SpecialInfo;             /* Matches struct Gadget */
    UWORD GadgetID;               /* Matches struct Gadget */
    APTR UserData;                /* Matches struct Gadget */

    /* These fields only exist under V39 and only if GFLG_EXTENDED is set */
    ULONG MoreFlags;
    WORD BoundsLeftEdge;        /* Bounding extent for gadget, valid */
    WORD BoundsTopEdge;         /* only if the GMORE_BOUNDS bit in   */
                                /*  the MoreFlags field is set.  The */
    WORD BoundsWidth;           /* GFLG_RELxxx flags affect these    */
    WORD BoundsHeight;          /* coordinates as well.              */
};
```

Intuition discerns a Gadget from an ExtGadget by testing the Flags field. If the GFLG_EXTENDED bit is set, the Gadget structure is really an ExtGadget structure. Intuition will use the bounding box only if the GMORE_BOUNDS bit in the ExtGadget.MoreFlags field is set.

Gadgetclass supports an attribute called GA_Bounds that sets up a Boopsi gadget's bounding box. It expects a pointer to an IBox structure (from *<intuition/intuition.h>*) in the ti_Data field, which gadgetclass copies into the bounding box fields in the ExtGadget structure.

Gadget Help

Intuition V39 introduces a help system designed around Intuition gadgets. Window-based applications can elect to receive a new type of IDCMP message called IDCMP_GADGETHELP when the user positions the pointer over one of the window's gadgets. The pointer is over the gadget if the pointer is within the gadget's bounding box (which defaults to the gadget's hit box).

Using the intuition.library function HelpControl(), an application can turn Gadget Help on and off for a window:

```
VOID HelpControl(struct Window *my_win, ULONG help_flags);
```

Currently, the only flag defined for the help_flags field is HC_GADGETHELP (from *<intuition/intuition.h>*). If the HC_GADGETHELP bit is set, HelpControl() turns on help for my_win, otherwise HelpControl() turns off gadget help for the window.

When gadget help is on for the active window, Intuition sends IDCMP_GADGETHELP messages to the window's IDCMP port. Each time the user moves the pointer from one gadget to another, Intuition sends an IDCMP_GADGETHELP message about the new gadget. Intuition also sends an IDCMP_GADGETHELP message when the user positions the pointer over a gadgetless portion of the window, and when the user moves the pointer outside of the window.

An application can find out what caused the IDCMP_GADGETHELP message by first examining the IntuiMessage's IAddress field. If IAddress is NULL, the pointer is outside of the window. If IAddress is the window address, the pointer is inside of the window, but not over any gadget. If IAddress is some other value, the user positioned the pointer over one of the window's gadgets and IAddress is the address of that gadget.

To discern between the different system gadgets (window drag bar, window close gadget, etc.) an application has to check the GadgetType field of the gadget:

```
#define GTYP_SYSTYPEMASK 0xF0   /* This constant did not appear in */
                                /* earlier V39 include files.      */

sysgtype = ((struct Gadget *)imsg->IAddress)->GadgetType & GTYP_SYSTYPEMASK;
```

The constant GTYP_SYSTYPEMASK (defined in <*intuition/intuition.h*>) corresponds to the bits of the GadgetType field used for the system gadget type. There are several possible values for sysgtype (defined in <*intuition/intuition.h*>):

GTYP_SIZING	Window sizing gadget
GTYP_WDRAGGING	Window drag bar
GTYP_WUPFRONT	Window depth gadget
GTYP_WDOWNBACK	Window zoom gadget
GTYP_CLOSE	Window close gadget

If sysgtype is zero, IAddress is the address of an application's gadget.

Not all gadget's will trigger an IDCMP_GADGETHELP event. Intuition will send gadget help events when the pointer is over a gadget with an extended Gadget structure (struct ExtGadget from <*intuition/intuition.h*>) and a set GMORE_GADGETHELP flag (from the ExtGadget.MoreFlags field).

To set or clear this flag for a Boopsi gadget, an application can use the GA_GadgetHelp attribute. Setting GA_GadgetHelp to TRUE sets the flags and setting it to FALSE clears the flag. Only the OM_SET method applies to this gadgetclass attribute.

To support gadget help, gadgetclass gained a new method called GM_HELPTEST. This method uses the same message structure as GM_HITTEST, struct gpHitTest (defined in <*intuition/gadgetclass.h*>), and operates similarly to GM_HITTEST. While a window is in help mode, when the user positions the pointer within the bounding box of a Boopsi gadget that supports gadget help, Intuition sends the gadget a GM_HELPTEST message.

Like the GM_HITTEST method, the GM_HELPTEST method asks a gadget if a point is within the gadget's bounds. Like the GM_HITTEST method, the GM_HELPTEST method allows a Boopsi gadget to have a non-rectangular hit area (or in this case, a help area). If the point is within the gadget, the gadget uses one of two return codes. If the gadget returns a value of GMR_HELPHIT, Intuition places a value of 0xFFFF in the Code field of the IDCMP_GADGETHELP message.