Amiga Mail
Volume II
January/February 1993

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

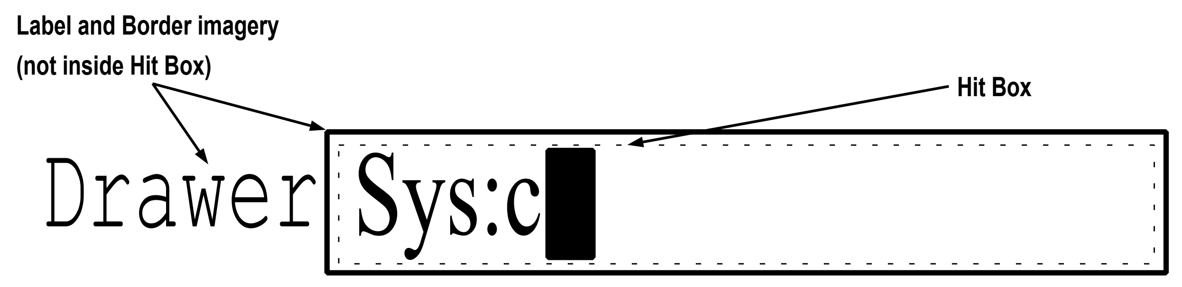

Figure 1 - String Gadget without Bounding Box

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.

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. The gadget also has the option of using GMR_HELPCODE instead of GMR_HELPHIT. GMR_HELPCODE is a little peculiar as a return value. Although GMR_HELPCODE is 32 bits long, Intuition identifies GMR_HELPCODE using only its upper 16 bits. If the upper 16 bits of GM_HELPTEST's return value matches the upper 16 bits of GMR_HELPCODE, Intuition copies the lower word of the return value into the Code field of the IDCMP_GADGETHELP message. The Boopsi gadget is free to set the return value's lower word to any 16-bit value.

If the point is not within the gadget's ``help spot'', the gadget returns GMR_NOHELPHIT. Intuition will then look under that gadget for more gadgets. If a gadget's dispatcher passes the GM_HELPTEST method on to the gadgetclass dispatcher, the gadgetclass dispatcher will always return GMR_HELPHIT.

An application can put several windows in the same *help group*. This feature groups several windows so that as long as one of those windows is active, the application can receive IDCMP_GADGETHELP messages about all of those windows. For more information, See the HelpControl() Autodoc and the WA_HelpGroup section of the OpenWindow() Autodoc (both are in *intuition.doc*).

Boopsi Gadgets and ScrollRaster()

This section covers some fairly complicated aspects of Intuition Windows and how Intuition interacts with the Layers system. For a more in depth explanation, see the article, ``Optimized Window Refreshing'' from the July/August 1992 issue of *Amiga Mail*.

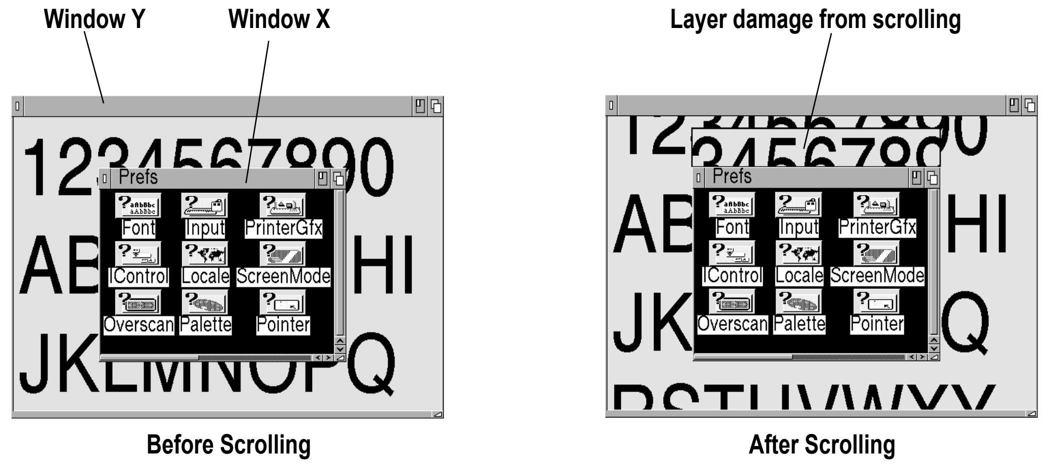

Figure 2 - Damage from a Scrolling Operation

Scrolling an Intuition display with ScrollRaster() is unlike many other rendering operations because it can damage a window layer. ScrollRaster() is both a rendering function and a layering function (ScrollRaster() is a *graphics.library* function, but it is aware of the Layers system). If window X overlaps window Y, scrolling window Y can scroll a portion of window Y out from underneath window X:

What is Layer Damage?

The Layers definition of ``damage'' is a little confusing. Looking at the illustration, two areas of window Y need to be refreshed: the area that used to be underneath window X, and the area that was scrolled into view at the bottom of window Y (the partially visible ``RSTUVWXY''). Layers considers only one of these areas to be ``damaged'', the area the was scrolled out from underneath window X. Layers considers it damaged because the damage was caused by the presence of another window layer. The task that called ScrollRaster() has no idea that there is an overlapping window layer, so it is up to Layers to account for that damage. On the other hand, the task that called ScrollRaster() knows that the area at the bottom of window Y needs refreshing, so the task knows to fix that area of the window. To Layers, damage is the area of a layer that needs refreshing as a result of layers

For both smart refresh and super bitmap windows, this does not present a problem. Layers remembers what was underneath window X and restores that portion of the display when a layers operation reveals it. When a scrolling operation damages a smart refresh or super bitmap window, Layers repairs the display.

For a simple refresh window, Layers remembers which portion of the window layer is damaged. Layers does nothing to repair the damage. Layers leaves repairing the damage to Intuition. After performing a layers operation on a window (such as scrolling a portion of a window), Intuition must check if that operation caused layer damage (either to that window or other windows on the display).

In Release 2, when Intuition told a scrolling Boopsi gadget to render itself, Intuition did not check for layer damage. If that gadget damaged the display with the ScrollRaster() function, Intuition did not repair that damage.

As of Release 3.0, Intuition has a flag set aside in the MoreFlags field of the ExtGadget structure called GMORE_SCROLLRASTER (defined in <*intuition/intuition.h*>). If this flag is set, when the gadget damages the display with ScrollRaster() (or ScrollRasterBF()), Intuition redraws *all* GMORE_SCROLLRASTER gadgets in the damaged window. The class dispatcher should take care of setting this flag when creating the gadget in the OM_NEW method.

ScrollRaster() is not the only function in Release 3 that can scroll a window's contents. For Release 3, the Intuition library gained a function specifically to scroll a window's raster. That function, ScrollWindowRaster(), is similar to ScrollRaster(), but ScrollWindowRaster() is

Intuition-friendly. Boopsi gadgets must not use this function, they should use ScrollRaster() or ScrollRasterBF() instead. Also any applications that use GMORE_SCROLLRASTER **gadgets must use ScrollWindowRaster() for scrolling. Note that gadgets supplied by third parties or Commodore may be** GMORE_SCROLLRASTER **gadgets. For more information on ScrollWindowRaster(), see its Autodoc.**

Smart Layout Gadgets

Prior to Release 3, Intuition took care of laying out GREL gadgets whenever the dimensions of the window change. This was a limitation for a GREL Boopsi gadget that wants to perform its own layout. Because Intuition didn't tell the Boopsi gadget that the window's dimensions changed, the gadget cannot react to the change.

As of Release 3, Intuition uses a new gadget method called GM_LAYOUT **to tell a GREL Boopsi gadget that its window's dimensions have changed. The** GM_LAYOUT **method uses the gpLayout structure (defined in <*intuition/gadgetclass.h*>) as its message:**

```
struct gpLayout
{
    ULONG               MethodID;
    struct GadgetInfo  *gpl_GInfo;
    ULONG              gpl_Initial; /* This field is non-zero if this method was invoked
                                     * during AddGList() or OpenWindow().  zero if this
                                     * method was invoked during window resizing.
                                     */
};
```

For GREL gadgets, Intuition sends a GM_LAYOUT **message just after erasing the gadget's bounding box. Intuition does not touch the values of the gadget's bounding box or hit box, it lets the gadget handle recalculating these values. Thee gadget can lay itself out based on the new window (or requester) dimensions found in the GadgetInfo structure passed in the** GM_LAYOUT **message (gpl_GInfo from the gpLayout structure). Intuition also adds the old and new bounding box of the gadget to the window's damaged regions so that the gadget will appear is its new position. The gadget must not perform any rendering inside the** GM_LAYOUT **method. Intuition will send a** GM_RENDER **message to the gadget when its time to redraw the gadget in its new position.**

There are two cases where Intuition sends a GM_LAYOUT **message:**

When the gadget's window is resized
When Intuition adds the gadget to a window

There are two ways for a window to gain gadgets. An application can explicitly add gadgets

using the AddGList() function. The window can also gain gadgets during it initialization in the OpenWindow() call.

For most GREL Boopsi gadgets, Intuition expects the values in the LeftEdge, TopEdge, Width, and Height fields to follow the existing convention for regular GREL gadgets. Each of these fields has a flag in the Gadget.Flags field (GFLG_RELRIGHT**,** GFLG_RELBOTTOM**,** GFLG_RELWIDTH**, and** GFLG_RELHEIGHT**, respectively). If that field's flag is set, Intuition expects the value in that field to be relative to either the window border or the window dimensions. For example, if** GFLG_RELRIGHT **is set, Intuition expects the value in the gadget's LeftEdge field to be relative to the right window border.**

There is a special kind of GREL Boopsi gadget called a *custom relativity* gadget. For this type of gadget, Intuition expects the values in the LeftEdge, TopEdge, Width, and Height fields to be absolute measurements, just like the values Intuition expects in these fields for non-GREL gadgets.

Setting the GFLG_RELSPECIAL **bit in the Gadget.Flags field marks the gadget as a *custom relativity* gadget. The best way to set this bit is by setting the** GA_RelSpecial **attribute to** TRUE **when creating the gadget.**

Disabled Imagery

Certain gadget types support using a Boopsi Image as its imagery. These gadget types include the non-Boopsi boolean gadget, the frbuttonclass gadget, and the buttongclass gadget (This can also include private gadget classes that support the GA_Image **attribute). The Gadget structure contains a field called GadgetRender which can point to a Boopsi Image object (the** GFLG_GADGIMAGE **bit in the Gadget.Flags field should also be set, see the``Intuition Gadgets'' chapter of the *RKRM:Libraries* for more information). Intuition uses this image as the gadget's imagery.**

When a gadget uses this scheme for its imagery, Intuition sends IM_DRAW **(and** IM_DRAWFRAME**) messages to the image object. These imageclass methods not only tell a Boopsi image to render itself, they also tell the image which state to draw itself. Some examples of image states include normal, selected, and disabled.**

Prior to Release 3, when Intuition sent a draw message to one of these images, it only used the normal and selected states. If the image was disabled, Intuition told the image to draw itself as normal or selected and would render the standard ``ghosted'' imagery over the gadget.

Under Release 3, Intuition will use the disabled state as well, but only if the gadget's image

object supports it. When Intuition adds the gadget to a window, Intuition tests the IA_SupportsDisabled **attribute of the gadget's image object. This imageclass attribute was introduced in Release 3. If this attribute is** TRUE**, the image supports disabled imagery (both the** IDS_DISABLED **and** IDS_SELECTEDDISABLED **states), so Intuition sets the** GFLG_IMAGEDISABLE **flag in the Gadget structure's Flags field. Intuition uses this flag to tell if it should use the image's disabled state in the** IM_DRAW/IM_DRAWFRAME **message.**

As the IA_SupportsDisabled **attribute is a fixed prope**
using the OM_GET **method. An application cannot**

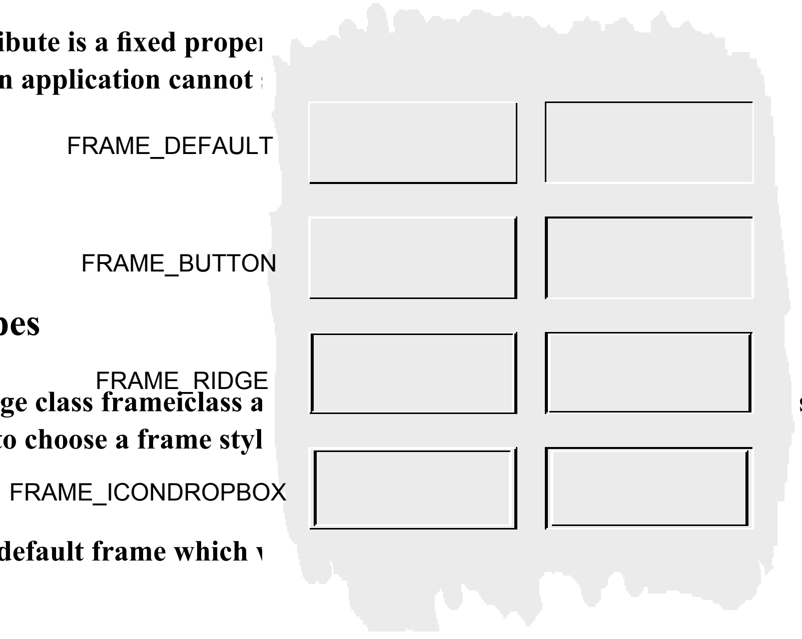

Figure 3 - frameiclass frame styles

Frameiclass Frame Types

For Release 3, the Boopsi image class frameiclass a **s**
attribute allows applications to choose a frame styl
four styles available:

FRAME_DEFAULT **- This is the default frame which**
thin edges.

FRAME_BUTTON **- This is the imagery for the standard button gadget. It has thicker sides and nicely edged corners.**

FRAME_RIDGE **- This is a ridge commonly used by standard string gadgets. When recessed (using the frameiclass attribute** IA_Recessed**), this image appears as a groove. Applications and utilities commonly use the recessed image to visually group objects on the display.**

FRAME_ICONDROPBOX **- This broad ridge is the system standard imagery for icon ``drop boxes'' inside of an AppWindow (see the ``Workbench and Icon Library'' chapter of the** ***RKRM:Libraries*** **for more information on AppWindows). The recessed version has no standard purpose.**

Additions to Sysiclass

The class of system images, sysiclass, gained two new images for Release 3, a menu check mark image and an Amiga key image (respectively, MENUCHECK **and** AMIGAKEY **from** ***<intuition/imageclass.h>*****):**

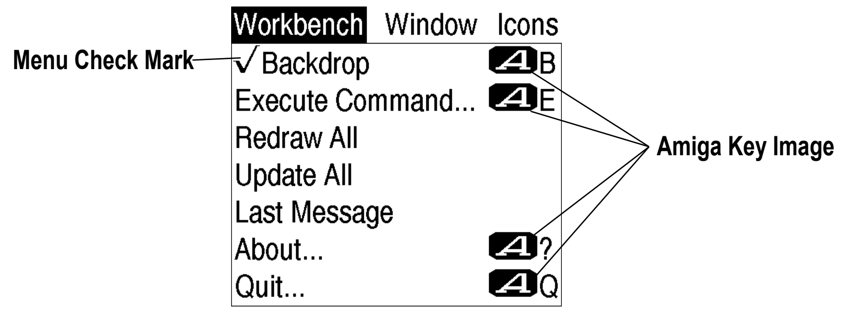

Figure 4 - Menu Check Mark and Amiga Key Images

This class also gained a new attribute called SYSIA_ReferenceFont**. This attribute, which is only accessible with the** OM_NEW **method, points to a TextFont structure. The** MENUCHECK **and** AMIGAKEY **images will use this font to figure out how large to make the image. This attribute overrides the** SYSIA_DrawInfo **font and** SYSIA_Size **attributes when calculating the dimensions of the image.**

About the Example

The example at the end of this article, ***relative.c*****, uses three features mentioned in this article. The example implements a private subclass of gadgetclass. The new class utilizes the frameiclass** IA_FrameType **attribute to create a button gadget. The class also takes advantage of the custom relativity feature. The example opens a window placing one of these gadgets in the middle of the window, sizing the gadget relative to the window's dimensions.**

The example also illustrates the gadget help feature of Intuition. When the private class dispatcher receives a GM_HELPTEST **message, it returns the bitwise AND of** GMR_HELPCODE **and the value 0xC0DE. When the example receives an** IDCMP_GADGETHELP **message at the window's message port, the example examines the message to find out what object triggered the help message and printf()s the results.**