Gadget

COLLABORATORS			
	TITLE : Gadget		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 19, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Gadget

1.1 Implementation notes

```
                          The Gadget classes
                  ------------------------
                  ($Date: 1994/05/08 12:57:58 $)
```

Intuition® Gadgets are wrapped up in the A++ Gadget classes, which are founded in the GadgetCV base class and specialized in the StdGadget, BoopsiGadget and GT_Gadget classes. GadgetCV is derived from GraphicObject (and therefore an IntuiObject).
What makes a Gadget being different from other graphical IntuiObjects is that it can react to user input that is, it receives IDCMP messages through its virtual 'callback' method. The GadgetCV class is the bridge-head to the GWindow class: all access to the GWindow in which the Gadget resides is channeled through the GadgetCV methods.

(In the following text 'gadget' refers to an Intuition® gadget while 'Gadget' with a capitalized 'G' refers to an A++ GadgetCV-derived object.)

How do GadgetCV objects receive user-input

How do GadgetCV objects fit into the GraphicObject sizing

How to derive own GadgetCV classes

-> Back to the root menu..

1.2 gadget_1

```
                GadgetCV auto-sizing
              --------------------------
```

The GadgetCV classes use the GraphicObject::redrawSelf() method to adjust the incorporated Intuition® gadget to the GraphicObject's new dimensions.

The 'redrawSelf()' method is being called everytime the window the gadget resides in has been resized. At the time of call the GraphicObject has already adjusted its dimensions. Now the GadgetCV object has to make its adjustments to the new dimensions.
GadgetCV objects are expected to return their internal Gadget structure address and the type of Gadget they incorporate.

The StdGadget, BoopsiGadget and GT_Gadget classes already have implemented their special redrawSelf methods to adjust their Intuition® gadget to the new dimensions.

1.3 gadget_2

The GadgetCV callback mechanism

The callback mechanism implemented in GWindow and GadgetCV objects handles all user activities on Gadget objects within one window.
The resulting IDCMP events are relayed to the respective GadgetCV object where they invoke the Gadget's callback method:

```
virtual void callback(const IntuiMessageC *imsg);
```

Now, the object can act on the message according to its class behaviour. At least, all changed internal attributes have to be propagated to the IntuiObject, which then can trigger notifications, with the call of

```
setAttrs( AttrList(MYCLASS_Tag1,newValue1,MYCLASS_Tag5,new5,..,TAG_END));
```

To protect the received IDCMP message, a new class has been declared above the 'IntuiMessage' that regulates access to it - the
IntuiMessageC class
.
The kind of IDCMP-Message-classes partially depends on the type of Gadget, but some messages are common to all Gadgets, as there are:

CLASS_GADGETDOWN - sent to the gadget that has become active, either by clicking (only when GACT_IMMEDIATE TRUE) or when being fall-back active gadget (see below).
CLASS_GADGETUP - sent to the active gadget when becoming inactive, either by the user releasing the mouse button (only when GACT_RELVERIFY) or when another gadget has become active.

The GWindow class manages an 'Active Gadget' to know where to send the messages that do not yield their origin that is, that have no reference to the Intuition® gadget which causes them (i.e. CLASS_MOUSEMOVE for non-GT gadgets; some classes like CLASS_RAWKEY never bear a gadget reference since they are not caused by a gadget!). In an object-oriented GUI library this is quite a problem since every user-event should be traced back to an object that reacts to it.
Now, the solution in A++ is, that only messages that name their gadget origin can change the Active Gadget to that origin while subsequent messages without identifier are directed to the last identified gadget, the Active Gadget.

Except for Gadtools gadget messages, only CLASS_GADGETDOWN and CLASS_GADGETUP

name their origin. So, if you have a Gadget that sends interim messages without address (like CLASS_MOUSEMOVE for Standard- and Boopsi-Propgadgets) set GACT_IMMEDIATE, so that GWindow can set Active Gadget and the messages are not being misdirected.

"Fine", you may say, "but which gadget is active on programm start?" - "Good question!". Why, You may choose your favorite Gadget to be the first active. Remember, the Active Gadget is the one on which the 'callback' method is invoked with each IDCMP message received.
Before entering the application loop, announce the first gadget to become active with

```
GWindow_Derived *myWindow;
GadgetCV_Derived *myFavoriteGadget;
myWindow->setActiveGadget(myFavoriteGadget);
```

If you do not choose one, no Gadget will be active on entry, and the user has to click on one to activate it.

There are three ways the current Active Gadget can loose its position:

1. its Intuition® gadget sends a CLASS_GADGETUP message.
   This CLASS_GADGETUP message, it can overrule to stay active, with

   ```
   BOOL GadgetCV::forceActiveGadget()
   ```

   returning TRUE to confirm the Active status.

2. another Intuition® gadget sends a CLASS_GADGETUP message: A++ considers the current Active Gadget having lost active status involuntarily and therefore reserves it to become Active Gadget again after the overruling Gadget has retreated from being the active one.
   In this case, the overruled Gadget will receive a faked CLASS_GADGETUP message when loosing Active Status, and afterwards will receive a CLASS_GADGETDOWN message when becoming active again.
   Both messages are empty except for the class identifier! You can detect such a message with a call to 'BOOL IntuiMessageC::isFakeMsg()' which will return TRUE on a message that did not originate from Intuition®.
   More useful, in case the Gadget forces to stay active, it can determine wether its request has been granted ('forceActiveGadget()' returns TRUE) or not.

3. you can yourself decide a Gadget to become the Active Gadget with a call to 'GWindow::setActiveGadget(GadgetCV *newActive)'. The current Active Gadget will receive a faked CLASS_GADGETUP message as described in (2).

See also: setting IntuiObject attributes

1.4 gadget_3

The GraphicObject base class declares the virtual redrawSelf() method. For basic introduction to its purpose click here.

Now the GadgetCV class uses this method in a more specific way (actually it

was designed to serve the GadgetCV class) to handle resizing of its gadgets:
The GadgetCV derived Gadget classes have to implement their special
redrawSelf() method that applies the GraphicObject new dimensions to the
Intuition® gadget.
GadgetCV derived objects are expected to return the address of their internal
gadget structure and the type of Intuition® gadget they are encaspulating.

Everything described here is already implemented for all types of Intuition®
gadgets in the StdGadget, BoopsiGadget and GT_Gadget classes.

1.5 gadget_4

The IntuiMessageC class

..makes you forget about the IntuiMessage structure members and gives
you some easy recognized methods that protect you and your program from
tampering with IntuiMessages.

First, all IDCMP message classes have an equivalent in the enumeration type
'IDCMPClass', where the prefix 'IDCMP_' has been exchanged for 'CLASS_',
thus, for instance, 'IDCMP_GADGETDOWN' becoming 'CLASS_GADGETDOWN'.

The IntuiMessageC inherits the IntuiMessage structure publically. So, all
structure members can be read (IntuiMessage comes as 'const' to your
'handleIntuiMsg()' method). But, for reason of code readability, some
methods have been added as there are:

```
IDCMPClass getClass() const;  // returns the IDCMP message class
APTR getIAddress() const;   // returns any object embedded in IAdress
```

The following ones may return NULL if it was not possible to gain the
information:

```
WindowCV *decodeWindowCV() const;   // get the message emanating window ( ↩
    safe on all messages)
GadgetCV *decodeGadgetCV() const;   // get the sending GadgetCV object (safe ↩
    )
```

Usually, you will only need to use 'getClass', the other methods are used
within GWindow class.

1.6 gadget_5

How to derive one's own GadgetCV class

First, remember that Gadget objects largely rely on the IntuiObject class.
Especially have a look at how the get/setAttributes methods are implemented
correctly (-> Class Implementor Notes).

Now, consider the

GadgetCV callback mechanism
.

Last, but not least, you have to write a redrawSelf() method for your Gadget class. The purpose your redrawSelf() method has to fulfill is to adjust your Gadget to the new Gadget dimensions, registered in the RectObject base class. Within the RectObject's inner boundaries you can virtually do everything that contributes to visualizing your special Gadget.

If you like to draw something, consider the AutoDrawArea class.

If you want to enhance a certain kind of Standard-, Boopsi- or GadTools-Gadget, have a look at the respective chapter.

If you need to group several Gadgets of any GadgetCV-derived class, read the chapter about the
Group Gadgets
.

1.7 gadget_6