Window

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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Window

1.1 Implementation notes

```
                              The Window classes
                        ------------------------
                        ($Date: 1994/05/08 12:58:50 $)
```

Intuition® windows are wrapped by the A++ Window classes which are based in the virtual WindowCV class. The GWindow class serves as working basis for your special application window class. It manages any GadgetCV-derived class objects you add to it. Interface graphics are refreshed properly, and user input is being distributed to the respective GUI objects.

(In the following text 'window' refers to an Intuition® window
while 'Window' with a capitalized 'W' refers to an A++ Window object.)

-> Back to the root menu..

1.2 window_1

```
          Receiving IDCMP messages from user-input
        ------------------------------------------
```

The WindowCV class sets up an IDCMP port to listen to all incoming messages from Intuition® that where triggered by the user acting on the window or its contents. This IDCMP port (exactly: the window user port) may be shared between several Windows.

To let a Window participate in another Window's user port, use the following Attribute Tag on the participating Window's constructor taglist:

```
MyWindow *window;   // MyWindow class is derived from GWindow
new MyWindow(OWNER, AttrList( ...,...,
      // note: this #define expands to a tag + value
      WCV_SharePortWithWindowObj(window), // type checking is provided
      ...,...,
      TAG_END) );
```

The destructor of a Window sharing its user port with other windows detaches itself safely from the user port. Only when the last Window is destructed, the user port is deleted.

The window is ready for receiving IDCMP messages directly after creation. IDCMP messages are represented in A++ by the IntuiMessageC class. Each incoming IDCMP message is delivered to the virtual method

```
virtual void WindowCV::handleIntuiMsg(const IntuiMessageC *imsg).
```

To achieve your own message handling, first, derive your Window class from GWindow and overwrite the method above, but make sure to invoke the root class' 'handleIntuiMsg()' method within your method. This is imperative for the GWindow class to maintain its task of dispatching IDCMP messages to the GadgetCV-derived objects within the window.

A possible Window class could look like this:

```
class MyWindow : public GWindow
{
   private:
      void On_CLOSEWINDOW()
      {
      }
   protected:
      void handleIntuiMsg(const IntuiMessageC *imsg)
      {
         switch (imsg->getClass())
         {
            case CLASS_CLOSEWINDOW : On_CLOSEWINDOW(imsg); break;
         }
         // propagate _EVERY_ message to the root class
         GWindow::handleIntuiMsg(imsg);
      }
   public:
      MyWindow(IntuiObject *owner,AttrList& attrs) : GWindow(owner,attrs)
      {
      }
};
```

1.3 window_2

```
      The auto resizing/refreshing
    ---------------------------------
```

When a window changes its size, due to user activation or programmed, its

```
GraphicObject base class gets an 'adjustChilds()' call (that is,
'adjustChilds()' is being called on the GWindow).
The 'adjustChilds()' method subsequently calls 'adjustChilds()' on each
child GraphicObject where it adjusts the GraphicObject's coordinates
according to its GOB_xxx Attribute Tags.
After all childs have been adjusted, first, all borders/backgrounds are
drawn (see GBorder class), then each child is requested to redraw itself
(the GraphicObject::redrawSelf() method is invoked). GadgetCV-derived
classes will reestablish their gadget(s) and return them to the GWindow,
other classes will draw themselves somehow.
(->redrawSelf() for class implementors)

This, and how GraphicObject constraints can be customized, is explained in
detail in the chapter about the GraphicObject class.
```

1.4 window_3

```
                Intuition® Fonts in a GWindow
              ----------------------------------

A font reference can be obtained with the FontC class.
Open a font, either ROM or disk font, with one of the following constructors:

   FontC(UBYTE *fontName=0,UWORD ySize=0,UBYTE style=0,UBYTE flags=0);
   // defaults to Preferences Screen Font.

   FontC(struct TextAttr *ta);   // open from an initialised TextAttr struct.
   FontC(struct TextFont *tf);   // open from an obtained TextFont structure
   FontC(const FontC& from);     // duplicate a FontC object

A FontC object can be used where a TextFont or a TextAttr structure is
requested.

The GWindow class provides the following methods to read the default fonts
set up in Preferences..

   const FontC& getScreenFont() { return screenFont; }
   // this is the Screen Text font from Preferences

   const FontC& getDefaultFont() { return defaultFont; }
   // and this the System Default font from Preferences
```