GraphicObject

COLLABORATORS			
	TITLE : GraphicObject		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 6, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

GraphicObject

1.1 Implementation notes

```
                The GraphicObject class
             ----------------------------
                ($Date: 1994/07/17 11:40:08 $)
```

The GraphicObject class plays the main role beside the IntuiObject class from which it is derived directly. It is used for all objects that have graphical dimension like windows, gadgets or draw areas, and it manages their geometry in dependency to other GraphicObjects. The relations stated in the IntuiObjects tree are used to express graphical relations: now each Graphic-object is placed and sized relative to its parent GraphicObject.
Each GraphicObject has to implement a redrawSelf() method to get automatic refresh after window sizing.

```
 How to position a GraphicObject
 The redrawSelf() method for class implementors
 How to decorate any GraphicObject with a customized border
 How to read the position and dimensions of a GraphicObject
 How to implement own graphical constraints (GOB_Tags)

------------------------------------------------------------
 -> Back to the root menu..
```

1.2 graphicobject_1

```
                GraphicObject positioning within the window
             ------------------------------------------------
                             (GOB_Tags)
```

GraphicObjects are positioned relatively. The position is specified with several Attribute Tags at creation time and may be changed during the object's lifetime with setAttributes().

The dimensions of one GraphicObject can be defined as being constant by usage of the GOB_Width/GOB_Height tags. Only two additional edges (Left,

Right edge and Top, Bottom edge) need to be determined, one edge for each orientation (horizontal/vertical).

Another way is to make the GraphicObject rectangle position itself relative to one edge of another GraphicObject. The GraphicObject then will possibly vary in its size depending on the movements of its edges.
Each edge can be attached to one of the two edges with the same orientation of either the parent GraphicObject or the predecessing sister GraphicObject. So, Left can be attached to Left or Right of either the parent or predecessor as can Top be attached to Top or Bottom of either the parent or predecessor. The same goes along with Right and Bottom.

The resulting tag can be constructed from combinations of these three groups:

Orient.:	Specify edge:	Specify related edge:		Specify related GraphicObject:
Horiz.	Left	Left		
	Right	Right		Parent
		From	Of	
Vert.	Top	Top		Pred (for 'predecessor')
	Bottom	Bottom		

(There you have (2*2*2) tags per orientation * 2 = 16 tags. Note, that some combinations are unresolveable. Add a 'GOB_' in front. See below.)

Examples:

1. 'GOB_LeftFromLeftOfParent' defines the distance of the left edge to the left edge of the parent to be the tag value.
2. 'GOB_TopFromBottomOfPred' defines the top edge to be placed in a distance of tag value pixels below (positive value) or above (negative value) the bottom edge of the predecessing GraphicObject.

The tag values (data) given for the tag are signed. The signum gives the direction from the view of the related edge where the specified edge is to be found, i.e. 'LeftFromLeftOfParent' places the left edge for positive values to the right of the left edge of the parent, 'LeftFromRightOfPred' places the left edge for negative values to the left of the right edge of the predecessor.
So, positive values place the specified edge right/below the related edge while negative values place the specified edge left/above the related edge.

Note, that no negotiations take place between a parent and its child about geometry (Like it is been known under OSF Motif®).
Geometry is passed on from parent to child descending trough the GraphicObject tree (which is in fact the IntuiObject tree).

The essential idea in GraphicObject class is of placing graphical boxes within other boxes and relative to some boxes. This is very unlikely to be changed in future versions of A++, while the way in which Attribute Tags specify these dependencies may easily be redesigned to meet the needs of the programmer.

1.3 graphicobject_2

The redrawSelf() method for class implementors

The redrawSelf() method is being declared virtual in GraphicObject as:

```
virtual ULONG redrawSelf(GWindow *homeWindow,ULONG& returnType);
```

It is called from the GWindow 'homeWindow' on all child GraphicObjects after the window size has changed and after the GraphicObjects have been adjusted to their new dimensions. The return value, further specified by 'returnType', has a special purpose for GadgetCV objects.

Derived classes can overwrite this method especially when they have own work to do to adjust themselves to their new GraphicObject dimensions. Other than GadgetCV-derived class should set 'returnType' to 0 and return NULL.

NOTE: Within your redrawSelf() method you must call redrawSelf() of your GraphicObject-derived base class! The redrawSelf() invokation is then propagated to the GraphicObject::redrawSelf() method, where borders and backgrounds are drawn (GBorder class). If you draw anything before having invoked redrawSelf(), your drawings could possibly be overdrawn from the border/background drawings!

If you derive your class from a GadgetCV-derived class, you must preserve the 'returnType' and return value of the base class' redrawSelf() call and return both from your redrawSelf() method!

The RectObject class, from which GraphicObject is derived, manages the Rastport coordinates of a GraphicObject within its owning GWindow.
If you want to draw inside your GraphicObject class, you need not to derive your class from GraphicObject but from DrawArea. DrawArea class provides draw methods that take coordinates relative to GraphicObject inner rectangle upper left edge and also clips any drawings that go beyond the inner rectangle.

1.4 graphicobject_3

The GBorder class

It is often appropriate to enhance any kind of GraphicObject, like Gadgets or DrawAreas, with a border drawing around its rectangle.
The GBorder class is a virtual base class that reserves room for a border in the GraphicObject by decreasing the GraphicObject inner dimensions (iLeft(), iTop(),iRight(),iBottom(),iWidth(),iHeight()), transparent to the GraphicObjects-derived classes and the class-user-defined GraphicObject position (via GOB_Tags).

There are two virtual methods that can be customized. The first one lets you define the dimensions of your special border class:

```
virtual void makeBorder(GraphicObject *graphicObj)=0;
```

Use RectObject::setBorders(leftBorder,topBorder,rightBorder,bottomBorder) to

tell GraphicObject the amount by which it has to diminish its rectangle.

The second method is called when the GraphicObject needs redrawing, but before any drawing is done. That allows a GBorder-derived class to not only draw in the border between outer and inner GraphicObject rectangle, but also in the background of the GraphicObject:

```
virtual void drawBorder(GraphicObject *graphicObj,GWindow *homeWindow);
```

Use the 'homeWindow' DrawArea methods to draw your border or background. Notice, that, when applying the GraphicObject coordinates (iLeft(),left(), iTop(),top(),...) to the DrawArea methods, these coordinates have to be transformed from Rastport coordinates to DrawArea coordinates first! Add an embracing NORM_X(gob_X_coord) or NORM_Y(gob_Y_coord) to the DrawArea method arguments.

1.5 graphicobject_4

The RectObject class

The RectObject class manages the dimensions of one rectangular box. It consists of an outer and an inner box, between these boxes lays the border area. The RectObject coordinates always are window's RastPort coordinates.

The following methods get the respective values from a RectObject:

```
XYVAL left(),top(),right(),bottom();
// get the RastPort coordinates of the outer bounding box

XYVAL iLeft(),iTop(),iRight(),iBottom();
// get the RastPort coordinates of the inner bounding box

WHVAL width(),height();
// get the pixel dimensions of the outer bounding box

WHVAL iWidth(),iHeight();
// get the pixel dimensions of the inner bounding box

WHVAL leftB(),topB(),rightB(),bottomB();
// get the dimensions of the border area
```

1.6 graphicobject_5

How to define own GOB_Tags

The GOB_xxx attribute tags describe how a GraphicObject is to be positioned within its home window in relation to other GraphicObjects.
To adjust each GraphicObject to a new window size, the GraphicObject has to read its GOB_xxx tags and compute the coordinates of its upper left and its

lower right inner box corners.

The virtual method

```
virtual void adjustChilds();
```

can be overwritten from derived GWindow and GadgetCV classes to implement a totally new way of interpreting GOB_xxx tags.

Now, let's have a look at what happens when a GWindow adopts to a changed window frame:

The window that has been resized gets an initial 'adjustChilds()' call from GWindow class. Its own RectObject values have already been adjusted and it then will have to step through its list of child GraphicObjects, computing the new position of each child from the GOB_xxx tag specifications.

Note, that each GraphicObject is fed with its RectObject coordinates by its parent GraphicObject. Therefore, a GraphicObject-derived class that overwrites 'adjustChilds()' implements new GOB_xxx tags only for its childs.

Within your 'adjustChilds()' method you can step through the childs in a simple FOREACHOF loop. Make sure to call 'adjustChilds() on each child you encounter, after you have adjusted its coordinates. You can read the GOB_xxx tags with an AttrIterator.

P.S.: If you have figured out a revolutionary new way of managing graphical constraints with GOB_xxx tags, let me know, so we may make it become standard for the GraphicObject class.