XEROX
INFORMATION PRODUCTS GROUP
Systems Development Division
July 30, 1977

To: Interested Persons

From: Bob Ayers

Subject: How "Frames" are Incorporated into Desktop

Stored: desktop-gfs.bravo

The mechanism which supports the "frame" of graphics imbedded in the desktop document can allow for the addition of different types of "frame" in a reasonable fashion. This memo is a overview of the frame mechanism which may help to indicate to the reader how he might arrange for his favorite object, say formulae, to exist within imbedded frames of their own.

The Desktop Object

Each desktop document, that is each of the windows that are showing a document that has been copied to the desktop, is defined by an instance of the record ShowInfoType. This is a long (the definition occupies almost exactly one page of paper) record which contains a lot of data about the desktop object that is not of general interest, such as the string that is in the "cut" wastebasket and the odometer's multiplicative factor. Two items within this record, however, are of general interest:

wh: WindowHandle is the MESA system pointer to its window object

gf: GFHandle [a pointer] is the beginning of a chain of frames that are in this document.

The GFObject

The gf field in the ShowInfoType record points to a GFObject, which is defined as follows:

```
GFObject: TYPE = RECORD [
        next: GFHandle,
        rptr: Rptr,
        x,y, w,h: INTEGER
        upFudge, rightFudge: INTEGER,
        edge: BOOLEAN,
        xfers: POINTER to something ]
```

Most of the fields within a GFObject serve to define the frame and its placement within the document. The xfers field serves to define the content of the frame and is a private pointer to the data appropriate to the particular stuff within the frame. (For graphics frames, the xfers field is the head of a chain of records, each of which defines one transfer symbol.)

Exact meanings of the fields:

next is the chain continuation to the next GFObject

rptr points to a system Rectangle object which defines the area of the bitmap that the frame now occupies. The normal use of this field is to *temporarily* put it into wh.rectangle and then use normal system window-painting functions to paint stuff into the frame.

x,y, w,h are the document-relative coordinates and dimensions of the frame. They have not been corrected for edges of the window or anything else.

upFudge and rightFudge handle a special case as follows: it may happen that the rectangle defined by rptr doesn't have the same coordinate origin as the frame -- that is point [0,0] in the rectangle may not be point [0,0] of the frame. This happens, for instance, if part of the frame is scrolled off the top of the window. [rightFudge,upFudge] are the frame coordinates of the [0,0] point of the rptr. [In the current desktop implementation, rightFudge is always zero -- I think.]

edge is a boolean which indicates whether the frame should have a line around its border.

Putting a New Frame Type Into the Desktop

I plan on introducing an enumerated type into the GFObject which will indicate which kind of frame it describes. I also plan on introducing a procedure variable into the GFObject which will be [a ShowInfoHandle is a pointer to a ShowInfoType]:

```
paintProc: PROCEDURE [ShowInfoHandle, GFHandle]
```

The responsibility of this procedure is to darken the bits in the frame that it wants dark. It can assume that, when it is called, the frame either has been cleared or it has the same bits that were painted the previous time -- this later case occurs on an upward scroll.

Depending on the sort of things that happen in the frame, there needs to be an interface which can track the cursor within the frame and accept typing when the cursor is in the frame. I plan on adding two other procedure variables to the GFObject:

mouseProc: PROCEDURE [ShowInfoHandle, GFHandle, x,y: INTEGER, b1,b2,b3: BOOLEAN]

typinProc: PROCEDURE [ShowInfoHandle, GFHandle, CHARACTER]

The mouseProc procedure will be called repeatedly whenever the cursor in in the frame. The typeinProc will be called whenever characters are typed to the frame. Note that the code which implements the frame cannot "track" the mouse by retaining control while the cursor is within the frame; it must keep returning from its mouseProc. This means, among other things, that it is impossible for the implementation to discover when the cursor leaves its frame.

Storage Requirements

Initially, the desktop will assume that all of the storage needs of the new frame mechanism are met through the storage the the "xfers" pointer points to. Actually, it may be convenient, later, to define new fields in the ShowInfoType to hold document-specific (rather than frame-specific) info that the new frame type requires.

Menus

Currently there is only one menu for the whole desktop document, and it has as many commands in it as will reasonably fit. We will have to discuss multiple menus and how they might work.

Cursor Shape

Currently the changing of the shape of the cursor to reflect, e.g., the "copy" function when the ctrl key is down, is done in an ad hoc fashion. If a new frame mechanism has many requirements on cursor shapes, then it would pay to design some sort of global scheme. If the mechanism can live with the current arrangement involving the shift and ctrl keys then we can leave it alone.

Frame Creation

Currently, a graphics frame with default size and no contents is created in response to the typein (into the document text) of a controlF. An extension of this mechanism to additional frame types (using additional special characters) may be appropriate.

Filing

It is the responsibility of a new type of frame to define a format for its data when the data is out on a file. This data should begin with some sort of unambiguous signal (the graphics stuff uses a controlG), and should be human-readable in the filed form -- so that it can be checked and possibly edited.

The new frame mechanism must supply a routine which can take a stream pointing to its filed-format data and construct its GFObjects. It must also supply a routine which can deliver a stream when a desktop document is being filed away.

The current graphic frame mechanism may serve as a (fairly ugly) example of how the above can be accomplished.