IMAGE OBJECTS

INTRODUCING OBJECTS INTO DOCUMENTS

Image Objects is a general-purpose interface that enables you to include graphics in documents. More generally, it enables you to include one kind of image, with its own semantics, layout rules, and editing paradigms, inside another kind of image. One side can be implemented by image users who want to include other images, and the other side can be implemented by image producers, who create images for use, say, in documents.

Images are encapsulated inside a uniform barrier--the IMAGEOBJ data type. From the outside, you communicate to the image by calling a standard set of functions. For example, calling one function tells you how big the image is; calling another causes the image object to be displayed where you tell it, and so on. Anyone who wants to create images for general use can implement his own brand of IMAGEOBJ. IMAGEOBJs have already been implemented for menus, bit maps, annotations, and graphs.

Similarly, any imaging system that wants to use other kinds of images has only to implement the consumer side of the interface and needn't worry about internal details.

Image Objects for *TEdit* allows objects to exist in *TEdit* documents and be edited with their own editor. It also provides a facility in which objects can be shift-selected (or "copy-selected") between *TEdit* and non-*TEdit* windows. For example, the *Image Objects* interface allows you to copy-select graphs from a *Grapher* window into a *TEdit* window. You can also perform more complicated copy-selections, such as copy-selecting text from a *DEdit* window into a *TEdit* window to preserve its structure, while copy-selecting it into a typescript window to treat it as a type-in. The source window (where the object comes from) does not have to know what sort of window the destination window (where the object is inserted) is, and the destination does not have to know where the insertion comes from.

A new data type, IMAGEOBJ, contains the data and the procedures necessary to manipulate an object that is to be part of a document. IMAGEOBJs are created with the function IMAGEOBJCREATE.

Another new data type, IMAGEFNS, is a vector of the procedures necessary to define the behavior of a type of IMAGEOBJ. Grouping the operations in a separate data type allows multiple instances

of the same type of image object to share procedure vectors. The data and procedure fields of an IMAGEOBJ have a uniform interface through the function IMAGEOBJPROP. IMAGEFNS are created with the function IMAGEFNSCREATE.

(IMAGEFNSCREATE *DisplayFn ImageBoxFn PutFn GetFn CopyFn ButtonEventInFn CopyButtonEventInFn WhenMovedFn WhenInsertedFn WhenDeletedFn WhenCopiedFn WhenOperatedOnFn PrePrintFn*) [Function]

returns an IMAGEFNS that contains the functions necessary to define the behavior of an IMAGEOBJ. The arguments are as follows:

DisplayFn is applied to the arguments *ImageObj* and *ImageStream.* It is called to display the object at the current position on *ImageStream.* The type of stream indicates whether the device is the display or some other image stream.

ImageBoxFn is applied to the arguments *ImageObj, ImageStream, CurrentX,* and *RightMargin.* It returns the size of the object as a list . The list's first and second elements are the width and height of the object image; its third and fourth elements are the position of the left edge and baseline of the image relative to where you want to position it. For characters, the third element is the descent (height of the descender) and the fourth element is zero because *TEdit* doesn't support left kerning. The *ImageBoxFn* returns the RECORD IMAGEBOX, which is a data structure that describes the image laid down when an IMAGEOBJ is displayed in terms of width, height, and descender height. It is provided with four fields: XSIZE, YSIZE, YDESC, and XKERN. The *ImageBoxFn* looks at the type of the stream to determine the output device if the object's size changes from device to device. (For example, a bit-map object may specify a scale factor that is ignored when the bit map is displayed on the screen.) The *CurrentX* and *RightMargin* fields allow an object to take account of its environment when deciding how big it is. If these fields are not available, they are NIL.

PutFn is applied to the arguments *ImageObj* and *FileStream.* It is called to save the object on a file. It prints a description on *FileStream* that, when read by *GetFn* (see below) , regenerates the *TEdit* object. (*TEdit* takes care of writing out the name of the *GetFn*.)

GetFn is applied to the arguments *FileStream* and *TextStream.* It is called when the object is encountered on the file during input. It reads the description that was written by the *PutFn* and returns an IMAGEOBJ.

CopyFn is applied to the argument *ImageObj* and returns a copy of *ImageObj.* The *CopyFn* is called during a copy-select operation.

ButtonEventInFn is applied to the arguments *ImageObj, WindowStream, Selection, RelX, RelY, Window, TextStream,* and *Button.* B*uttonEventInFn* is called when you press a mouse button inside the object. The *ButtonEventInFn* decides whether or not to handle the button, to track the cursor in parallel with mouse movement, and to invoke selections or edits supported by the object (but see *CopyButtonEventInFn* below). If *ButtonEventInFn* returns NIL, *TEdit* treats the button press as a selection at its level. Note that when this function is first called, a button is down. The *ButtonEventInFn* should also support the button-down protocol to descend inside of any composite objects with in it. In most cases, the *ButtonEventInFn* relinquishes control (i.e., RETURNs) when the cursor leaves its object's region.

CopyButtonEventInFn is applied to the arguments *ImageObj* and *WindowStream.* It is called when you button inside an object and hold down a copy (shift) key. Many of the comments about *ButtonEventInFn* apply here too. Also, see the discussion below about copy-selecting objects.

WhenMovedFn is applied to the arguments *ImageObj, TargetWindowStream, SourceTextStream,* and *TargetTextStream.* It provides hooks by which the object is notified when *TEdit* performs an operation (MOVEing) on the whole object. It allows objects to have side effects.

WhenInsertedFn is applied to the arguments *ImageObj, TargetWindowStream, SourceTextStream, and TargetTextStream.* It provides hooks by which the object is notified when *TEdit* performs an operation (INSERTing) on the whole object. It allows objects to have side effects.

WhenDeletedFn is applied to the arguments *ImageObj, TargetWindowStream, SourceTextStream,* and *TargetTextStream.* It provides hooks by which the object is notified when *TEdit* performs an operation (DELETEing) on the whole object. It allows objects to have side effects.

WhenCopiedFn is applied to the arguments *ImageObj, TargetWindowStream, SourceTextStream,* and *TargetTextStream.* It provides hooks by which the object is notified when *TEdit* performs an operation (COPYing) on the whole object. *WhenCopiedFn* is called in addition to (and after) the *CopyFn* above. It allows objects to have side effects.

WhenOperatedOnFn is applied to the arguments *ImageObj, WindowStream, HowOperatedOn, Selection,* and *TextStream.* It provides a hook for edit operations. The values of *HowOperatedOn* are SELECTED, DESELECTED, HIGHLIGHTED, and UNHILIGHTED. The *WhenOperatedOnFn* differs from the *ButtonEventInFn* because it is called when you extend a selection through the object. That is, the object is treated *in toto* as a TEdit character. HIGHLIGHTED refers to the selection being highlighted on the screen, and UNHIGHLIGHTED means that the highlighting is being turned off.

PrePrintFn is applied to the argument *ImageObj.* It is called to convert the object into something that can be printed for inclusion in documents. It returns an object that the receiving window can print (using either PRIN1 or PRIN2,its choice) to obtain a character representation of the object. If *PrePrintFn* is NIL, the OBJECTDATUM itself is used. TEdit uses this function when you indicate that you want to print the characters from an object rather than the object itself (presumably using PRIN1 case).

IDENTIFYING LEGITIMATE GETFNS

Each legitimate *GetFn* needs to be known to the system, to prevent various Trojan-horse problems and to allow the automatic loading of the supporting code for infrequently used IMAGEOBJs. To this end, there is a global list, IMAGEOBJGETFNs, that contains an entry for each *GetFn.* The existence of the entry marks the *GetFn* as legitimate; the entry itself is a property list (PList), which can hold information about the *GetFn* (see the following section).

No action needs to be taken for *GetFns* that are currently in use: the function IMAGEFNSCREATE automatically adds its *GetFn* argument to the list. However, packages that support obsolete versions of objects may need to explicitly add the obsolete *GetFns.* For example, TEdit supports bit-map IMAGEOBJs. Recently, a change was made in the format in which objects are stored; to retain compatibility with the old object format, there are now two *GetFns.* The current *GetFn* is automatically on the list, courtesy of IMAGEFNSCREATE. However, the code file that supports the old bit-map objects contains the clause: (ADDVARS (IMAGEOBJGETFNS *(OldGetFnName)*)), which adds the old *GetFn* to IMAGEOBJGETFNS.

For a given *GetFn,* the entry on IMAGEOBJGETFNS may be a property list of information. Currently the only possible property is FILE.

FILE is the name of the file that can be LOADed if the *GetFn* isn't defined. This file also defines all the other functions needed to support that kind of IMAGEOBJ.

For example, the bit-map IMAGEOBJs implemented by *TEdit* use the GETFN BMOBJ.GETFN2. Its entry is (BMOBJ.GETFN2 FILE IMAGEOBJ), indicating that the support code for bit-map image objects resides on the file IMAGEOBJ, and that the *GetFn* for them is BMOBJ.GETFN2.

This makes it possible to have entries for *GetFn*s whose supporting code isn't loaded--you might, for instance, have your init file add entries to IMAGEOBJGETFNS for the kinds of image objects you commonly use. The system's default reading method will automatically load the code when necessary.

USING THE BUILT-IN SUPPORT FUNCTIONS

(IMAGEFNSP *X*) [Function]

returns *X* if X is an IMAGEFNS, NIL otherwise.

(IMAGEOBJCREATE *ObjectDatum ImageFns*) [Function]

returns an IMAGEOBJ that contains the object datum *ObjectDatum* and the operations vector IMAGEFNS. *ObjectDatum* can be arbitrary data.

(IMAGEOBJP *X*) [Function]

returns *X* if *X* is an IMAGEOBJ, NIL otherwise.

(IMAGEOBJPROP *ImageObject Property NewValue*) [Function]

accesses and sets the properties of an IMAGEOBJ. It can be used on the system properties OBJECTDATUM, DISPLAYFN, IMAGEBOXFN, PUTFN, GETFN, COPYFN, BUTTONEVENTINFN, COPYBUTTONEVENTINFN, WHENOPERATEDONFN, and PREPRINTFN. Additionally, it can be used to save arbitrary properties on an IMAGEOBJ.

READING AND WRITING IMAGEOBJS ON FILES

IMAGEOBJs can be written out to files using HPRINT and read back using READ. This method uses the following functions, which can also be used in user code:

(WRITEIMAGEOBJ I*mageObj Stream*) [Function]

PRIN2s a call to READIMAGEOBJ, then calls the PUTFN for *ImageObj* to write it onto *Stream*. During input, then, the call to READIMAGEOBJ is read and evaluated; it in turn reads back the object's description, using the appropriate *GetFn.*

(READIMAGEOBJ *Stream GetFn NoError*) [Function]

reads an IMAGEOBJ from *Stream*, starting at the current file position. Uses the function *GetFn* after validating it (and loading support code, if necessary). If the *GetFn* can't be validated or isn't defined, READIMAGEOBJ returns an IMAGEOBJ that is a safe encapsulation: It displays as a rectangle that says, "Unknown IMAGEOBJ Type" and lists the *GetFn's* name. However, if NOERROR is non-NIL, READIMAGEOBJ returns NIL if it can't successfully read the object.

Selecting an encapsulated IMAGEOBJ with the mouse causes another attempt to read the object from the file; this is so you can load any necessary support code and then get to the object. NB: You cannot save an encapsulated IMAGEOBJ on a file because there isn't enough information to allow copying the description to the new file from the old one.

COPY-SELECTING BETWEEN WINDOWS

The general idea behind copying between windows is that the source window builds an image object of what you have selected and calls a system function. The system function finds the current TTY window and calls a function associated with it that knows how to insert image objects. For instance, BKSYSBUF is used to copy strings by having the system function call BKSYSBUF on the PREPRINTFN of the image object if the target window doesn't have an insert function.

The following new functions have been added to the system to implement copy-selecting image objects between windows.

COPYBUTTONEVENTFN [Window property]

is called (if it exists) when a button event occurs and a copy key is down. If no COPYBUTTONEVENTFN exists, the BUTTONEVENTFN is called.

COPYINSERTFN [Window property]

is called by COPYINSERT (see below) when the source window wants to insert something into the destination window as a result of a copy-select. Its arguments are the object to be inserted and the destination window. The object to be inserted can be a character string, an IMAGEOBJ, or a list of IMAGEOBJs and character strings. As a convention, COPYINSERTFNs call BKSYSBUF if the object they are to insert is a character string. For *TEdit* windows, the COPYINSERTFN is a function that calls TEDIT.INSERT if given a string and TEDIT.INSERT.OBJECT if given a *TEdit* object.

(COPYINSERT *ImageObject*) [Function]

is applied to the argument *ImageObject.* It inserts *ImageObject* into the window that currently has the TTY. It first finds the window that has the TTY (say *TTYWindow*). If *TTYWindow* has a COPYINSERTFN, this is called, passing it *ImageObject.* If no COPYINSERTFN exists and if *ImageObject* is an image object, the result of calling its PREPRINTFN on it is BKSYSBUFed; otherwise *ImageObject* is BKSYSBUFed. (The default BKSYSBUF uses PRINT2 with a read table taken from the process associated with the TTY window. A window that wishes to use PRIN1 or a different read table can provide its own COPYINSERTFN that does this.) Thus the

COPYBUTTONEVENTFN for a window allows you to select an object and then calls COPYINSERT on an image object built (via IMAGEOBJCREATE) from the selected object and 13 functions that define its behavior.

End Notes

1. *TEdit* calls the IMAGEBOXFN only during line formatting, then caches the imagebox as (IMAGEOBJPROP obj 'BOUNDBOX). This avoids the need to call IMAGEBOXFN when incomplete position and margin information is available.

2. When the *ButtonEventInFn* is called, the window's clipping region and offsets have been changed so that the lower-left corner of the object's image is at (0,0), and only the object's image can be changed. The selection is available for changing to fit your needs; the mouse button went down at (relX,relY) within the object's image. You can affect how *TEdit* treats the selection by returning one of several values. If you return NIL, *TEdit* forgets that you selected an object; if you return the atom DON'T, TEdit doesn't permit the selection; if you return the atom CHANGED, *TEdit* updates the screen. Use CHANGED to signal TEdit that the object has changed size or will have side effects on other parts of the screen image.

3. As with the *ButtonEventInFn* , the offset and clipping regions for the display are set so the object's image is at (0,0), and only that image area can be modified.