Introducing Objects into Documents

stored on: {phylum}<lispcore>library>Imageobj.TEdit
last changed: July 16, 1984
by: R. Burton J. Sybalsky

The following proposal arose out of discussions between John Sybalsky, Ron Kaplan and Richard Burton about putting objects into Tedit and integrating shift selection among various types of windows.

The goals are:

1) define an object interface for Tedit that allows objects to exist in a document and be editted with their own editor. (Currently for formatting, Tedit will view an object as a large character and not try to "flow" text around it. Maybe later.)

2) provide a facility in which objects could be shift selected (hereafter referred to as "copy selected" as in Star) between windows in which the source window (where the object was coming from) does not have to know what sort of window the destination window is and the destination does not have to know where the insertion came from. As a simple example, we want to be able to copy select from a Grapher window into a Tedit window. More complicated, we want copy select from a Dedit window into a Tedit window to preserve structure (if desired) while copy select into a typescript window to treat it as type-in.

Image Objects

To do this, we provide a new datatype, IMAGEOBJ, which contains the data and the procedures necessary to manipulate an object that is to be part of a document. Additionally, there is a new datatype, IMAGEFNS, which is a vector of the procedures necessary to define the behavior of a type of IMAGEOBJ. Having the operations grouped in a separate datatype allows multiple instances of the same type of image object to share their procedure vectors. The data and procedure fields of an IMAGEOBJ have a uniform interface through the function IMAGEOBJPROP. IMAGEOBJs are created with the function IMAGEOBJCREATE. IMAGEFNS are created with the function IMAGEFNSCREATE.

(IMAGEFNSCREATE DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN BUTTONEVENTINFN COPYBUTTONEVENTINFN WHENMOVEDFN WHENINSERTEDFN WHENDELETEDFN WHENCOPIEDFN WHENOPERATEDONFN PREPRINTFN)

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

DISPLAYFN - fn of {imageObj imageStream} called to display the object at the current position on imageStream - type of stream indicates whether device is DISPLAY, PRESS or INTERPRESS.

IMAGEBOXFN - fn of {imageObj imageStream currentX rightMargin} should return the size of the object as a list whose 1st and 2nd elements are the width and height of the object image and whose 3rd and 4th elements are the position of the left edge and baseline of the image relative to the current position. For characters, the 3rd element would be the descent and 4th element would be 0 since we don't support left kerning. We will also want to support 5th and 6th elements which are the amounts to move the current position in the X and Y directions after the object is displayed. The RECORD IMAGEBOX is provided with fields (XSIZE YSIZE YDESC XKERN). The IMAGEBOXFN looks at the type of the stream to determine the output device if the size changes from device to device. (For example, a bitmap object may specify a scale factor that is ignored when it is displayed on the screen.) {note: this does not address, for example, the laying out of annotations as inline text, footnotes or between line text.) The currentX and RightMargin fields allow an object to take account of its environment when deciding how big it is. If these are not available, they will be NIL.

TEDIT NOTE: TEdit only calls this function during line formatting, then caches the imagebox as (IMAGEOBJPROP obj 'BOUNDBOX). This avoids the need to call IMAGEBOXFN when incomplete X & margin info is available.

PUTFN - fn of {imageObj fileStream} called to save the object on a file. Prints characters or whatever on fileStream that when read by GETFN (see below) will regenerate the Tedit object. (Tedit takes care of writing out the name of the GETFN.)

GETFN - fn of {fileStream TextStream} called when the object is encountered on the file during input. It should read the stuff that was put out by the PUTFN and return an IMAGEOBJ. {note: we may want to have a library of image object types that is searched when a GETFN is encountered that is undefined which would load the information necessary to handle that type of object.)

COPYFN - fn of {imageObj} should return a copy of imageObj. The COPYFN is called during a copy-select operation.)

BUTTONEVENTINFN - fn of {imageObj windowStream Selection relX relY window TextStream Button}. The user has pressed a button inside the object. The BUTTONEVENTINFN should decide whether or not to do handle the button, track the cursor and respond to the button presses to bring about whatever edit (or selection but see COPYBUTTONEVENTINFN below) protocols the object wished to support. If BUTTONEVENTINFN returns NIL, TEdit will treat the button press as a selection at its level. Note that when it is first called, a button will be down. It is envisioned that Tedit itself could be used recursively to edit text that was within an object. Also the BUTTONEVENTINFN should support the button down protocol to descent inside of any composite objects with it. It is also envisioned that the BUTTONEVENTINFN will relinquish control (i.e. RETURN) when the cursor leaves its object's region although there may be cases where it would not. In any case, standard useful functions for cursor tracking and editting in the Tedit tradition will be made available to lessen the task of building BUTTONEVENTINFNs.)

TEDIT NOTE: When this function gets 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: Return NIL and TEdit will forget that you selected an object; return the atom DON'T and TEdit won't permit the selection; return the atom CHANGED, and TEdit will update the screen. Use this latter to signal TEdit that the object has changed size or should have side effects on other parts of the screen image.

COPYBUTTONEVENTINFN - fn of {imageObj windowStream}. The user has buttoned inside an object and a copy key is held down. (In initial implementations, the copy keys will be the shift keys. This is probably something that should be settable (in the terminal table?).) Many of the comments about BUTTONEVENTINFN apply here also. Also see the discussion below about copy-selecting objects below.

WHENINSERTEDFN -

WHENMOVEDFN -

WHENDELETEDFN -

WHENCOPIEDFN - fns of {imageObj TargetWindowStream SourceTextStream TargetTextStream}. Provide hooks by which the object can get notified when Tedit performs an operation on the whole object. The different operations are: INSERT, MOVE, DELETE, COPY. {Also need a hook for delete undone which may be the same as insert.} WHENCOPIEDFN will get called in addition to (and after) the COPYFN above. These functions allow objects to have side effects. For example, annotations in a documentation maintains a window

which has summary of all annotations in the document. These functions allows that summary to be updated as the annotation is operated on by Tedit.

WHENOPERATEDONFN - fn of {imageObj windowStream howOperatedOn Selection TextStream}. Provides a hook for miscellaneous edit operations. For now the values of howOperatedOn are SELECTED ~~and DESELECTED~~, HIGHLIGHTED and UNHILIGHTED. This is different from the BUTTONEVENTINFN because it will be called when the user is extending a selection through the object i.e. the object is being treated in toto as a character from Tedit. HIGHLIGHTED and UNHIGHLIGHTED refer to the selection being highlighted on the screen, and to having the highlighting turned off. These are called when the selection is inside the object, so it may handle it.

> *TEDIT NOTE: As with the BUTTONEVENTINFN, the offset and clipping region for the display are set so the object's image is at (0,0), and only that image area can be modified.*

PREPRINTFN - fn of {imageObj} called to convert the object into something that can be printed for inclusion in documents. It should return 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 would use this function when the user indicates that he wants the characters from an object rather than the object itself (presumably using PRIN1 case). (Interface to this "unstructure" operation is not yet determined.)

(IMAGEFNSP X) - Returns X if X is an IMAGEFNS, NIL otherwise.

(IMAGEOBJCREATE OBJECTDATUM IMAGEFNS)

Returns an IMAGEOBJ which contains the object datum OBJECTDATUM and the operations vector IMAGEFNS. OBJECTDATUM can be arbitrary data.

(IMAGEOBJP X) - Returns X if X is an IMAGEOBJ, NIL otherwise.

(IMAGEOBJPROP IMAGEOBJECT PROPERTY {NEWVALUE})

Accesses and sets 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.

Copy selecting between windows

The general idea behind copying between windows is that the source window builds an image object of what the users has 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. The simple case done now with BKSYSBUF to copy strings is done 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 things will be added to the system to implement copy selecting of things between windows.

A new window property COPYBUTTONEVENTFN will be called (if it exists) when a button event occurs and a copy key is down. If no COPYBUTTONEVENTFN exists, the BUTTONEVENTFN is called.

A new window property COPYINSERTFN will be called by COPYINSERT (see below) when another window wants to insert something into this window as a result of a copy select. Arguments are the thing to be inserted and this window and the thing to be inserted. The thing to be inserted can be (1) a STRINGP, (2) an IMAGEOBJ, or (3) a list of IMAGEOBJs and STRINGPs. As a convention, COPYINSERTFNs should call BKSYSBUF if the object they are to insert is a STRINGP. For Tedit windows, the COPYINSERTFN will be a function

that calls TEDIT.INSERT if given a string and calls TEDIT.INSERT.OBJECT if given a Tedit object. (John, can the current copy select between TEdit windows be changed to use this by having a case in the COPYINSERTFN that handles PIECEs?)

A new function (COPYINSERT imageObject) will insert imageObject into the window that currently has the TTY. It finds the window that has the tty (say ttyWindow). If ttyWindow has a COPYINSERTFN, this is called, passing it imageObject. If no COPYINSERTFN exists, if imageObject is an image Object, the result of calling its PREPRINTFN on it is BKSYSBUFed; otherwise imageObject is BKSYSBUFed. (The default BKSYSBUF will use PRINT2 with a read table taken from the process associated with the ttyWindow. A window which wishes to use PRIN1 or a different readtable can provide its own COPYINSERTFN that does this.) Thus the COPYBUTTONEVENTFN for a window should allow the user to select an object and then calls COPYINSERT on an image object built (via IMAGEOBJCREATE) from the selected object and nine functions that define its behavior.

Misc notes:

The layout and displaying function of TEdit will be made available for use within image object displayfns. Thus, an annotation could call the TEdit formatting routines to lay itself out within a box. {This actually also needs to determine the size of the box which should depend on the margins.}

In the case of Dedit copy selecting a list into a TEdit window, the displayfn could PRINTDEF the structure and possibly Dedit it after it was in the Tedit document. In this case the structure would need to have some idea of the width to be printed into (which should also be editable somehow). Possibly the structure that Tedit builds could have an extra field for width which it defaults upon creation but which could be changed. This case is interesting because it represents a case where we might want Tedit to pass information into the object (ie. how wide to print itself).

This does not address the problem of image objects that print as text. For example, annotations might be printed in line in a smaller font or as footnotes at the bottom of the page or as margin notes or between the lines intermixed with the main text. This is a difficult problem that is independent of the specification of this proposal.

Provided Image Object types

Menus. Supported by John Sybalsky.
Bitmaps - supported by Greg NuyensRichard Burton.
Annotations - supported by Richard Burton.

Future object types

Grapher graphs - supported by Ron Kaplan.
S-expressions - volunteers?