TKbutton

XFCN
No. 4470
Version: 5.0 (ONLY for Registered owners of ToolKit™)

TKbutton allows you to create or poll pseudo buttons that conform to any graphics shape. This XFCN stores a region of the selected graphics as a named resource into the stack. Calling TKbutton with the name (or list of names) will return the name of any button that is "hit".

∂

Examples

ÓÊget TKbutton("make","rat,cat,dog,elephant")
ÊÊ›get TKbutton("phit","rat,cat,dog,elephant",20)
ÊÊ›get TKbutton("hit","rat,cat,dog,elephant",-1)

get TKbutton("make","rat,cat,dog,elephant")

In this example it is assumed there are masks stored as PICT resources in the stack with the names "Rat", "Cat", etc. TKbutton will create a region resource for each of the PICTs.

get TKbutton("phit","rat,cat,dog,elephant",20)

This is slower but does not require the creation of region resources. TKbutton dynamically creates a region for each PICT and tests to see if the mouse is within the region. The 20 passed in the third parameter tells TKbutton to invert the first "button" hit and maintain the inversion for 20 60ths (1/3) of a second. You will receive the name of the FIRST "button" hit.

get TKbutton("hit","rat,cat,dog,elephant",-1)

This is fast because it uses region resources stored inside the stack. TKbutton tests to see if the mouse is within the region. The -1 passed in the third parameter tells TKbutton to invert the first "button" hit track the mouse, hiliting and unhiliting the "button" as appropriate. You will receive the name of the FIRST "button" hit.

Parameters

Parameter 1

name: name of PICT or region resource
This is the name of a PICT resource if you are using "make" or "phit" as the second parameter. If you are using "hit" as the second parameter then it is the name of a button region.

Parameter 2

make: used to create a button region from a PICT resource

phit: used to poll TKbutton to see if "hit", or mouseDown, occurs within the region of a PICT resource. This does NOT require a region resource in the stack. This is slower than a normal hit because the region has to be dynamically created from the PICT resource for sampling.

hit: used to poll TKbutton to see if a "hit", or mouseDown, occurs within a buttons region resource. This is fast since the region already exists, having been created with the make command.

Parameter 3

hilite: This is a number. If it is negative (-1) then the button is inverted and tracked (see "More Information" below). It hilite is zero (0) there is no hiliting or tracking. If hilite is positive the selected button will invert for the period indicated by the value of hilite. Positive numbers represent 60ths of a second.

How It Works

Understanding how TKbutton works will be important to using it efficiently. PICTs only have "shape" or form to the observer. The Macintosh sees a PICT as a series of instructions that tell it what to draw and how to draw it. Part of a PICT record is a rectangle defining the boundries of the PICT. There is nothing to indicate the limits of individual shapes within the PICT itself.

To define the components of a PICT we need something called a Region. Regions are a series of points that tell the Macintosh what the shape of a graphic is AND the coordinates of the shape. Let's use a picture example of a persons face. We may want to have buttons for each major component of the face, such as eyes, nose, ears, hair, mouth, etc.

With the old method we would have created a button to fit each item. Naturally, we would size the button to fit as tightly as possible. Since each button is defined as a rectangle it would be easy to select the nose, for example, even though we really clicked the mouse on the face. With TKbutton we can define the nose as a region so that a response will happen only if the mouse click occurs within the exact boundries of the nose.

TKbutton has the facilities to both create the region and to test the region for a mouse click.

Creating Button Regions

Let's look at the four key elements to creating regions:

. Create your graphics
. Place your graphics
. Create PICTs of the graphic components
. Use TKbutton to create the region of each PICT

1. Create your graphics

You can draw your graphics in HyperCard™ or in any graphics environment, like MacPaint, MacDraw, or Canvas, and import the graphics to your card or background. There is not much mystery about this step.

2. Place your graphics

Placing your graphics onto the card or background is another matter. There are two basic senarios. The first involves working only in the HyperCard™ environment. What is unique about HyperCard™ is if you lasso and copy any graphics on a card or background the coordinates of the graphics is maintained in the copy.

Suppose I draw a face in the lower right portion of the card. I then lasso the face and copy it using "Copy Picture..." in the edit menu of HyperCard™.

Ó his has effectively converted the graphics to a PICT which is now stored in the clipboard.

If I move to another card and paste that PICT, it will show up in the same location on the new card as where it was originally copied from. Likewise, if I use TKdoPICT or the ToolKit™ Utilities to store that PICT as a resource inside my stack, it will retain its position.

This is not necessarily true of PICTs created in other drawing environments which brings us to the second senario. Many drawing programs set the coordinates to zero, zero (the topleft corner of a window)! So do most screen capture utilities. This means if you create a graphic in a drawing environment other than

HyperCard™, copy the graphic, move to HyperCard™, and use the ToolKit™ Utilities to store what you have copied as a PICT resource in your stack it will (most likely) NOT have any positioning information with it. Even if you are carful to draw in the same relative position within your drawing program that you would expect the graphics to occur in the HyperCard™ window this information will probably be lost.

The way to overcome this is to take a little detour before converting the PICT in the clipboard to a PICT resource in your stack. Copy your graphic from your drawing environment, return to HyperCard™, and then:

. Go to the card you want the graphics on.
. Do a paste from the edit menu of HyperCard™.
. Position the graphics where you want it.
. Do a Copy Picture... from the edit menu to recapture the
/graphics along with its proper coordinates.
. Now you can create the PICT resource in your stack.

What we have discussed here pertains to maintaining positioning information of graphics. There are other considerations to creating the individual PICTs.

3. Create PICTs of the graphic components

Let's keep the example of a face and the components of a face (eyes, nose, ears, etc.). The best way to create the component PICTs is to work with two cards. The first card is the card that will contain the finished product. You want to keep this card. The second card is a scratch card which you will dispose of when you are done.

We will draw our face (or import the graphics) onto our card. Now we lasso the face, go to our scratch card, and paste the face. First, I want a PICT of just the head. To do this I will erase the ears, neck, and hair line. Now I must create a mask of the face.

What is a mask and how do I make one? Regions do not recognize "white" space. I want a button of the face so that if a mouse click occurs anywhere where there is no feature, like the nose or an eye, I want to tell the user they selected the face. If I just create a region of the face this will only work if the mouse click is on a portion of the drawing where the pixels are black, such as the outline of the face, some shadowing, or a wrinkle line. If the cheeks, for example, are white then no mouse click will be recognized.

This is easily overcome by copying the graphic and "filling" it with the black pattern using the "Fill" selection of the HyperCard™ Paint menu. For this to be successful you need to understand leakage.

The easiest way to understan leakage is to picture a circle. If I lasso the circle and Fill it with the black pattern I will have a black ball. However, if there is a hole in my circle, that is, a single pixel in the outline is missing (white) then the circle will not fill. This is called leakage. When I selected the circle with the hole in its outline I effectively selected curved line that almost, but not quite, formed a circle. The effect is the same as selecting a straight line. You cannot Fill a line!

In creating a mask I do not want any leakage. When I remove the ears, neck, and hair from the face I must make sure I have a complete outline of the face. In this case I will probably have to examine, in fat bits, the hair line to make sure I have no holes in the outline.

Once I am satisfied the outline is complete I can lasso it, Fill it, copy the filled mask, and store the result as a PICT resource in the stack with the name "Face".

The process is now the same for all the components of the face:

. Copy the original on the source card.
. Paste it onto the scratch card.
. Erase everything except the part I want a button of.
. Create a mask of the remaining graphics.
. Copy the mask.
. Store the copy as a PICT resource with an appropriate

/name ("Ears", etc.)

4. Use TKbutton to create the region of each PICT

Finally, we are ready to create the regions. What we have stored in the stack is a series of PICTs with names like "Face", "Ears", "Eyes", and "Nose". Use TKbutton to create the regions:

```
et TKbutton("make","Face,Ears,Eyes,Nose")
```

This converted ANY PICT resource in the stack with the names passed in the second parameter into resources of regions with the same names.

Using The "Buttons"

We have not created a button in the normal sense of a HyperCard™ button. You cannot just grab it, move it around, or open a script for the button. Using our region resources as buttons is quite simple:

```
et TKbutton("hit","Ears,Eyes,Nose,Face",-1)
```

The hit tells TKbutton to examin each name, if a region resource exists by that name then see if the mouseDown is within that region. The -1 passed in the third parameter tells TKbutton to invert the region if it is found and track the mouse movements, hiliting or unhiliting the region appropriately.

This call to TKbutton must be in a mouseDown handler which will reside in the card, background, or stack script.

Ó hat is returned to you is the name of the FIRST region encountered where the mouse is within the region.

In this example we passed the "Face" as the last name because if it were first we would never get a hit on the "Eyes" or "Nose". As soon as any region qualifies there is no more activity by TKbutton, other than tracking the "button" if requested. Here is an example script:

Script of Card "Face Anatomy"

```
on mouseDown
 et TKbutton("hit","Ears,Eyes,Nose,Mouth,Face",-1)
 f it does not = "" then
  f it = "Ears" then
   ut "You selected the ears"
  lse if it = "Eyes" then
   ut "You selected the eyes"
    •
    •
  lse if it = "Face" then
   ut "You selected the face"
  nd if
 nd if
 ass mouseDown
end mouseDown
```

Requirements

ÓÊThe PICT resource should be in the stack. If it is not present the only indication you will get is a Beep.

Return Values

ÓÊThe name of the first button "hit" is returned.

Related Topics

ÓÊSee TKdoPICT and the ToolKit™ Utilities for storing PICT resources into the stack from the clipboard.

ÓÊSee TKshowPICT for color.   See the discussion for using color below.

Special Features

ÓÊThe ToolKit™ Button Editor makes creating, destroying, and moving region "buttons" easy.   It should be evident to you by now that moving a region "button" is not easy.   In fact, you have to go through the entire creation process for each component of the graphics for the new location.

The ToolKit™ Button Editor allows you to select a region "button" from a menu and drag it on the screen, just like a real HyperCard™ button.

More Information

ÓÊUsing Color
Obviously, when creating graphics in the HyperCard™ drawing environment we have no color.   How, then, are we going to apply color to our region buttons?

First, treat everything as though it were black and white to create the regions.

Next, save a PICT resource of the entire picture, in color, in your stack.   Copy your graphics from another drawing program or use a screen capture utility to grab what you want into the clipboard.   Use TKdoPICT or the ToolKit™ Utilities to create a PICT resource.   In our example we will call it "Head".

When you open the card use TKshowPICT to display the color PICT exactly over the B&W graphics on the HyperCard™ card.   Finding the exact coordinates is not much problem.   Simply type into the message window:

  KshowPICT "Head",0,0

Hit the return key and the PICT should appear in the upper left corner of the window.   Now adjust the left and right coordinates and hit the return key again.   Keep doing this until the color PICT is aligned perfectly over your B&W graphics.   When you have it the way you want it just copy the statement from the message window and paste it into your openCard script.

Hiliting with TKbutton will not interrupt the color PICT that overlays it.   In fact, that portion of the color PICT will hilite!

°°