Getting Grabby with the QNX Photon microGUI

by John Fehr

At some point you might find that you need to take a snapshot of your display (or a window in your display) and process it in some way. There are many possible scenarios for accomplishing this. But in this article, you'll learn how to 'grab' the window that's currently under your mouse cursor, whether it's active or not.

Which window?

The first thing you'll have to figure out is which window is under your mouse cursor. This is where **PhQueryCursor**() comes in handy because, among other things, it tells you which region is currently under your cursor. The only problem with it is that it doesn't tell you the window's region, but rather the top-most region in that window. How do you figure out which window it belongs to?

PhRegionQuery() gives you quite a bit of information on any region you specify. Among other things, it tells you which flags are set for this region, and who its parent region is. How does this help you? Every region in an application's window is generally owned by that application. However, the first region created (usually a **PtWindow**) is inside a region owned by the Window Manager. (The Window Manager draws your windows frame, the buttons on the frame, etc., and manages them for you.) One of the bits in a region's flags is **Ph_WND_MGR_REGION**, which tells you that this region is owned by the Window Manager. So, if you keep asking for your region's parent until you get to the one that's owned by the Window Manager, you end up with the region of the window you're interested in!

Here's how you'll get the **PhRid_t** of the window under your cursor:

```
PhRid_t current_window(void)
{
  PhCursorInfo_t buf;
  PhRegion_t region;
  PhRect_t window;
  // grab the cursor's info
  PhQueryCursor(PhInputGroup(NULL),&buf);
  // get the region information of the region under the cursor
  PhRegionQuery(buf.region,&region,NULL,NULL,0);
  // search parent until we reach a window manager's region
  while (region.rid)
  {
    // return the window manager's region we found
    if (region.flags&Ph_WND_MGR_REGION) return region.rid;
    // this only happens if our window manager's not running
    if (PhRegionQuery(region.parent,&region,NULL,NULL,0)==-1)
      break;
  }
  // failed (window manager's not runnning)
  return 0;
}
```

Now that you have the window's region id you should be able to grab the window's contents, correct? Well, yes and no. All you can really grab is the area of the screen that corresponds to the window's rectangle. So,

if anything else is partially covering up the window you want to grab, you'll capture the covering portion as well.

Getting your window's rectangle

How do you get the window's onscreen rectangle? If you do a **PhRegionQuery**(), the rectangle you pass in will be set to the region's rectangle relative to its parent's region, not the screen's region. You can use **PhTranslateRect**() to translate your window's rectangle to its parent's region and then do the same for your parent's parent's region, etc.
Here's how you'll get the **PhRect_t** of the window/region you're interested in:

```
int windows_rect(PhRid_t rid,PhRect_t *r)
{
  PhRect_t window;
  PhRegion_t region;
  // grab the region for your rid
  if (PhRegionQuery(rid,&region,&window,NULL,0)==-1) return -1;
  // translate the regions rect to its parent until you hit the bottom
  while (region.rid)
  {
    PhTranslateRect(&window,&region.origin);
    if (PhRegionQuery(region.parent,&region,NULL,NULL,0)==-1)
      break;
  }
 // return the windows fully translated rect
  *r=window;
  return 0;
}
```

Take a picture, it'll last longer!

Finally, you now know which area of the screen you want to capture. How do you take the actual snapshot?

The QNX RTOS gives you a convenient pair of functions called **PgReadScreenSize**() and **PgReadScreen**(). **PgReadScreenSize**() tells you how much shared memory you'll need to allocate to hold the part of the display you're going to capture. (You need to use shared memory because the graphics driver has to access it.) Once you've allocated the shared memory (using **PgShmemCreate**()), you call the **PgReadScreen**() function to copy the portion of the screen you're interested in into your shared memory, and return a **PhImage_t** structure.

To save this image to your file, you simply call **PxWriteImage**() with your file name and image, and specify an output format (BMP format in this example).

Here's how you'll grab and save the window:

```
int snap(char *fname,PhRect_t rect)
{
  int len;
  char *mem;
  PhImage_t *image;
  // get the size of the shared memory you need for the image
  if ((len=PgReadScreenSize(&rect))<=0) return -1;
```

```
  // allocate shared memory to contain the image
  if (!(mem=(char*)PgShmemCreate(len,NULL))) return -1;
  // read the portion of the screen you're interested in
  if (image=PgReadScreen(&rect,mem))
  {
    // save the image to the file name given
    PxWriteImage(fname,image,NULL,PX_IMAGE_BMP,0);
  }
  // destroy the shared memory you used
  PgShmemDestroy(mem);
  return 0;
}
```

Putting it all together

OK, you've got all the functions you should need. Now you'll add a couple of header includes at the beginning:

```
#include <stdio.h>
#include <photon/PhT.h>
#include <photon/Pg.h>
// make sure we include BMP support!
#define PX_IMAGE_MODULES
#define PX_PHIMAGE_SUPPORT
#define PX_BMP_SUPPORT
#include <photon/PxImage.h>
```

What should you do to test this? Just capture the contents of the window under the cursor as fast as you can, 50 times. You'll add the following at the end of your source file:

```
int main(int argc,char *argv[])
{
  int i;
  if (argc!=2)
  {
    printf("Usage: grab <pattern>");
    exit(-1);
  }
  // get a connection to Photon
  if (PtInit("/dev/photon")!=0)
  {
    printf("Error: Could not connect to Photon.");
    exit(-1);
  }
  // grab and save the current window 50 times
  for (i=0;i<50;i++)
  {
    PhRid_t r=current_window();
    PhRect_t rect;
    char fname[120];
    windows_rect(r,&rect);
    sprintf(fname,argv[1],i);
    printf("saving '%s'\n",fname);
```

```
    snap(fname,rect);
  }
  return 0;
}
```

Testing it out

Now, let's test this puppy out. Assuming you've saved the file as *octopus.c*, you can create your octopus binary with:

```
qcc octopus.c -o octopus -lph -lphexlib
```

The file should compile and link without any warnings or errors. When it's finished, test it out by typing './octopus image%2.2d.bmp.' It will immediately start capturing the contents of the window under the cursor, so try moving the cursor around over various windows. When it's finished you should have 50 image*.bmp files in your current directory. You can view them with the **pv** application, (i.e. 'pv image02.bmp').

Happy grabbing!