Dragon Drop

Authored by: **John Fehr**
Updated by: **Mikhail Nefedov**

One of the least used and least understood features of the QNX Photon microGUI is its flexible method of drag and drop. In this article, we'll unravel some of the mysteries behind this very useful data transport method.

Any kind of data can be dragged and dropped, but there are a few standard types that we should use if we want other "drag 'n drop"-aware applications to be able to communicate with us. The most common is probably plain text, but image data is also often transmitted.

Aren't dragons a myth?

At the moment, there are very few QNX Photon applications that readily accept any kind of drag and drop information. So, in order to show you how this works, we'll have to run two copies of our sample program.

Dragon bible

The standard user method of starting a drag is by left clicking the mouse on our widget and then moving the mouse holding the mouse button down. If this is already used for some other functionality, then the drag should only be started while also holding the control key down. If possible, we should use this standard with any applications we want to drag information out of.

OK, let's write some code! Let's start with something simple - a little application (call it *dnd.c*) that contains a **PtList** widget, a **PtText** widget, and three little baby **PtButton** widgets. Each of the buttons will have slightly different checkered images. Here's our code:

```
#include <stdio.h>
#include <photon/PtProto.h>
#include <photon/PtWidget.h>
#include <photon/PtWindow.h>
#include <photon/PtList.h>
#include <photon/PtText.h>
#include <photon/PtButton.h>
// Create a checkerboard 16x16 image.
PhImage_t *create_checkerboard(short color1,short color2)
{
  PhImage_t *i;
  PhGC_t *ogc;
  char *bits;
  int a,b;
  i=PhCreateImage(NULL,16,16,Pg_IMAGE_DIRECT_565,NULL,0,1);
  bits=i->image;
  for (a=0;a<16;a++,bits+=i->bpl)
    for (b=0;b<16;b++)
      ((short*)bits)[b]=(((a/2)+(b/2))&1)?color1:color2;
  return i;
}
int main()
{
   PtArg_t args[5];
   PtWidget_t *win,*list,*text,*button1,*button2,*button3;
   PhImage_t *image1,*image2,*image3;
   PhDim_t dim={200,300};
   PhPoint_t pos={50,50};
   char *items[]={"one","two","three","four"};
   PtInit("/dev/photon");
   // create our window
   PtSetArg(&args[0],Pt_ARG_DIM,&dim,0);
   PtSetArg(&args[1],Pt_ARG_POS,&pos,0);
   PtSetArg(&args[2],Pt_ARG_WINDOW_TITLE,"Dragon Droppings",0);
   win=PtCreateWidget(PtWindow,Pt_NO_PARENT,3,args);
   // create our text widget
   dim.w=150;dim.h=25;pos.x=25;pos.y=200;

PtSetArg(&args[2],Pt_ARG_FLAGS,Pt_TRUE,Pt_SELECTABLE|Pt_SELECT_NOREDRAW
);
   text=PtCreateWidget(PtText,win,3,args);
   // create our list widget
   dim.w=dim.h=150;pos.x=pos.y=25;
   list=PtCreateWidget(PtList,win,3,args);
   // add a few items to our list widget
   PtListAddItems(list,(const char **)items,4,1);
   image1=create_checkerboard(0xff70,0x0000);
   image2=create_checkerboard(0x07ff,0xffff);
   image3=create_checkerboard(0x0000,0xffff);
   // create first button widgets
   dim.w=dim.h=26;pos.x=25;pos.y=250;
   PtSetArg(&args[2],Pt_ARG_LABEL_TYPE,Pt_IMAGE,0);
   PtSetArg(&args[3],Pt_ARG_LABEL_IMAGE,image1,0);
   button1=PtCreateWidget(PtButton,win,4,args);
   // create second button widget
   pos.x=150;
   PtSetArg(&args[3],Pt_ARG_LABEL_IMAGE,image2,0);
   button2=PtCreateWidget(PtButton,win,4,args);
   // create our third button widget
   pos.x=78;dim.w=50;
   PtSetArg(&args[3],Pt_ARG_LABEL_IMAGE,image3,0);
   button3=PtCreateWidget(PtButton,win,4,args);
   PtRealizeWidget(win);
   PtMainLoop();
 }
```

That seems straightforward enough. We added the **Pt_SELECTABLE|Pt_SELECT_NOREDRAW** flags to the text because we'll need them later. We can compile this with '`gcc dnd.c -o dnd -lph`' to try it out. Not too exciting, is it?

What a drag!

Why don't we add drag capability to our list widget? The best time to start a drag would be when the left mouse button is down and we move the mouse. The perfect solution would be to use a **Pt_CB_OUTBOUND** callback. Let's add the following code after the list creation function call:

```
PtAddCallback(list,Pt_CB_OUTBOUND,list_outbound,0);
```

We should also add some code that will initiate the drag in our **list_outbound**() function. To do this, we create a **PtTransportCtrl_t** object with the **PtCreateTransportCtrl**() function, add some simple inline data (i.e. the data is copied) with the **PtTransportType**() function, and start the dragging with the **PtInitDnd** call. (The **Pt_DND_SILENT** call tells the drag-and-drop routines not to inform the source widget of the drag- and-drop progress. We don't care about it after the drag and drop is started.) Here's the code to put in before the **main**() function:

```
  int list_outbound(PtWidget_t *widget,void *data,PtCallbackInfo_t
*cbinfo)
 {
   PhPointerEvent_t *pev=PhGetData(cbinfo->event);
   // make sure we have the left (selection) button pressed
   if (pev->buttons&Ph_BUTTON_SELECT)
   {
     unsigned short *ind;
     short *num;
     // get the list of indexes (there can be only one!)
     PtGetResource(widget,Pt_ARG_SELECTION_INDEXES,&ind,&num);
     // make sure there's a selected item
     if ((*num)>0)
     {
       char **items;
       PtTransportCtrl_t *tctrl=PtCreateTransportCtrl();
       // get the list of items
       PtGetResource(widget,Pt_ARG_ITEMS,&items,&num);
       // add an inline plain text transport to the drag and drop
transport
       // control, making sure we remember that indexes start at 1 in
the list
       PtTransportType(tctrl,"text","plain",0,Ph_TRANSPORT_INLINE,
         "string",items[ind[0]-1],0,0);
       // start the drag and drop
       PtInitDnd(tctrl,widget,cbinfo->event,NULL,Pt_DND_SILENT);
     }
   }
   return Pt_CONTINUE;
 }
```

Now, when we compile *dnd.c*, run it, and click and drag on the list widget, the cursor turns into a little circle with a slash through it. The cursor indicates where you can drag your information to, and since we don't have anything that accepts it, it'll stay as this "not dropable" cursor no matter where we move it.

Look out for dragon droppings!

OK, now that we've been able to initiate a drag, how about accepting a drop somewhere? Hmm... Which widget will we choose? How about the **PtText** widget? The **PtText** widget accepts plain text drag-and-drop by default. At last we can drag and drop something. Lookin' good! Try running multiple copies of **dnd**, and try dragging from one copy's list into another copy's text box. Pretty spiffy, eh?

What about just dragon?

What if we only want to drag an item inside our list? Luckily, there's some increased functionality provided for just that purpose. Let's start by adding a **Pt_CB_DND** callback for the list widget as well. Add the following line after the **Pt_CB_OUTBOUND PtAddCallback** function call for the list widget:

```
PtAddCallback(list,Pt_CB_DND,list_drop,0);
```

But what do we do in the **list_drop()** callback? Well, we do the same kind of thing that we did for the **PtText** drop callback's **Ph_EV_DND_ENTER** code, but our **Ph_EV_DND_DROP** code is a little different. We receive information as to where in the list our drag was dropped - the item, the item index (keeping in mind that the first item is at index 1), and whether the drop was before, after, or on, the given item.

All we really need to do, then, is figure out where to put the dropped text, making sure that if that text already exists somewhere in our list, we delete it first. One other difference in the list drop is that the **select_flags** member in the **PtDndFetch_t** array must have **Pt_DND_SELECT_MOTION** set. This will give us a visual indication of where we're dropping inside the list widget.

Here's the code to put in after the **text_drop** function:

```
int list_drop(PtWidget_t *widget,void *data,PtCallbackInfo_t *cbinfo)
{
  PtListDndCallback_t *ldndcb=cbinfo->cbdata;
  PtDndCallbackInfo_t *dndcb=&ldndcb->dnd_info;
  static PtDndFetch_t wanted_dnd[] =
  {
    {"text","plain",Ph_TRANSPORT_INLINE,Pt_DND_SELECT_MOTION },
  };
  int ind=ldndcb->item_pos-1; // the index of the item we're over
  switch (cbinfo->reason_subtype)
  {
    case Ph_EV_DND_ENTER:
      PtDndSelect(widget,wanted_dnd,ARRAY_SIZE(wanted_dnd),
        NULL,NULL,cbinfo);
      break;
    case Ph_EV_DND_DROP:
      if (ldndcb->flags&Pt_LIST_ITEM_DNDSELECTED_DOWN)
        ind++;
      // if the item exists already, delete the original
      if (PtListItemExists(widget,dndcb->data))
      {
        int oldpos=PtListItemPos(widget,dndcb->data);
        PtListDeleteItemPos(widget,1,oldpos);
        if (oldpos<ind) ind--;
      }
      PtListAddItems(widget,(const char**)&dndcb->data,1,ind+1);
      PhFreeTransportType(dndcb->data,"text");
      break;
  }
  return Pt_CONTINUE;
}
```

Now we can move our items inside our list or drag them into the text widget.

If we would like the drag to work only inside our list, and not be droppable elsewhere, we would set the **Pt_DND_LOCAL** flag in the **PtInitDnd**() call. This is really useful when you need to drag around private data, such as internal pointers that are meaningless to other applications within your application.

Checkered dragons

We've seen how text dragging and dropping works. How about other kinds of data? Let's try dragging images! (Bet you were wondering why we put in those buttons and those checkerboard images, eh?) It turns out that dragging images (and actually any kind of data) is very similar to dragging text. Let's start with the dragging part. We'll make the left and right buttons draggable, and we'll make them drag whatever image they hold. But first, we'll need to add a callback to each of those buttons. Add the following after the second button is created:

```
   PtAddCallback(button1,Pt_CB_OUTBOUND,button_outbound,0);
   PtAddCallback(button2,Pt_CB_OUTBOUND,button_outbound,0);
```

In this case, what we're doing is generic enough that the two buttons can share the same callback. That callback looks remarkably similar to the text outbound callback. Add the following just before main:

```
  int button_outbound(PtWidget_t *widget,void *data,PtCallbackInfo_t
*cbinfo)
 {
   PhPointerEvent_t *pev=PhGetData(cbinfo->event);
   // make sure we have the left (selection) button pressed
   if (pev->buttons&Ph_BUTTON_SELECT)
   {
     PhImage_t *image;
     PtTransportCtrl_t *tctrl=PtCreateTransportCtrl();
     PtGetResource(widget,Pt_ARG_LABEL_IMAGE,&image,0);
     // inline the button's image to the drag
     PtTransportType(tctrl,"image","an image",0,Ph_TRANSPORT_INLINE,
       "PhImage",image,0,0);
     // start the drag and drop
     PtInitDnd(tctrl,widget,cbinfo->event,NULL,Pt_DND_SILENT);
   }
   return Pt_CONTINUE;
 }
```

Now we're able to drag, but what about the drop? We'll let the middle button accept drops. (We could have just as easily made any of the other two buttons accept a drop as well.) Add the following after the third button is created:

```
 PtAddCallback(button3,Pt_CB_DND,button_drop,0);
```

Please note that we use a shallow **free**() instead of a deep **PhFreeTransport**() free. This is because a deep free would free the actual image pixels (which we don't want), since the button still uses them. Add the following code just before main:

```
 int button_drop(PtWidget_t *widget,void *data,PtCallbackInfo_t
*cbinfo)
 {
   PtDndCallbackInfo_t *dndcb=cbinfo->cbdata;
   static PtDndFetch_t wanted_dnd[] =
   {
     {"image",NULL,Ph_TRANSPORT_INLINE, },
};
   switch (cbinfo->reason_subtype)
   {
     case Ph_EV_DND_ENTER:
       PtDndSelect(widget,wanted_dnd,ARRAY_SIZE(wanted_dnd),
         NULL,NULL,cbinfo);
       break;
     case Ph_EV_DND_DROP:
       PtSetResource(widget,Pt_ARG_LABEL_IMAGE,dndcb->data,0);
       free(dndcb->data);
       break;
   }
   return Pt_CONTINUE;
 }
```

Woohoo! Works exactly like we wanted!

Mating dragons?

Let's add one complication. What if we wanted a widget to be able to accept either an image drop or a text drop? It turns out this is easier than you'd think. First, we add an extra line to our **wanted_dnd** fetch array in the **button_drop** function:

```
{"text","plain",Ph_TRANSPORT_INLINE, },
```

Now we're able to accept plain text drops, but how do we distinguish between the types of drops?

Again, it's easier than you'd think. The **PtDndCallbackInfo_t** structure has a member called **fetch_index**, which indicates which element in the fetch array is being dropped. Let's replace the original **Ph_EV_DND_DROP** case in the **button_drop** function with:

```
      case Ph_EV_DND_DROP:
       switch(dndcb->fetch_index)
       {
         case 0: // first index from fetch array, image
           PtSetResource(widget,Pt_ARG_LABEL_TYPE,Pt_IMAGE,0);
           PtSetResource(widget,Pt_ARG_LABEL_IMAGE,dndcb->data,0);
           free(dndcb->data);
           break;
         case 1: // second index, text
           PtSetResource(widget,Pt_ARG_LABEL_TYPE,Pt_Z_STRING,0);
           PtSetResource(widget,Pt_ARG_TEXT_STRING,
             dndcb->data,strlen(dndcb->data));
           PhFreeTransportType(dndcb->data,"text");
           break;
       }
       break;
```

This way, our original code is executed if we're dropping an image (making sure the widget is set to a **Pt_IMAGE** type label), and setting the widget's text if we're dropping a text string (making sure the widget is set to a **Pt_Z_STRING** type label)!

Congratulations. You are now an honorary dragon tamer! Build it, and they will come!