
Chapter 7 List Manager & LDEFS

List Manager advice.....	112
List Bummers Revisited (simple custom Ldef with code).....	112
List Bummers Revisited (and Debugging Help).....	114
List definitions and fonts.....	114
Cdevs and the List Manager.....	115

USENET Macintosh Programmer's Guide

From: jackiw@cs.swarthmore.edu (Nick Jackiw)

Subject: Re: List Manager advice

rdd@walt.cc.utexas.edu (Robert Dorsett) writes:

```
> I'm attempting to use the List Manager, from within a dialog, to implement a
> one-column list of text cells. The length of the list can vary, with items
> My problem is that I'm unable to get the cell ID of the currently-selected
> item. I'm convinced I'm adding data to the list correctly list correctly
> (dataBounds for the list looks okay), but if I try LLast to get the point, it
> doesn't work. If, on the other hand, I use LGetSelect, the numbers are too
> high (but not by much, which raises the maddening question of whether I'm
> installing the data correctly).
>
> Robert Dorsett
```

I assume from your post that your list permits only one item to be selected at a time. Good.

```
begin {Modal List Dialog}
theDLOG:=GetNewDialog(theDLOG_Id,nil,pointer(-1));
InstallUserItem(theDLOG,List_ITEM,@DrawList);
SetPort(theDLOG);

{At this point, it's assumed the list already exists; you've done LNew}

theList^^.SelFlags:=lOnlyOne;

ShowWindow(theDLOG);
LDoDraw(true,theList);

repeat
  ModalDialog(@ModalProc,itemHit);
  if itemHit=List_ITEM then           {Translate double-clicks into Opens}
    if LClick(GlobalPt,0,theList) then itemHit:=Open_ITEM;
  until (itemHit=Open_ITEM) or (itemHit=Cancel_ITEM);

  if itemHit=open_ITEM then
begin {Figure out which item is selected; that's what we want to open}
  SetPt(aPt,0,0);
  junk:=LGetSelect(true,aPt,theList);
  aStr:='#####';
  oldLen:=length(aStr);
  LGetCell(ptr(longint(@aStr)+1),oldLen,aPt,theList);

  {aStr now contains the text of your item to be opened; act appropriately}
  ...
end;
end;
```

This scheme uses a global variable, globalPt, to record the coordinates of the last mousedown (that's the purpose of modalProc), so that our main loop, above, can send that point to the list manager for double-clicks. (Alternately, doubleClicks could be detected in the filterProc, which would obviate the need for a global.)

If this doesn't help, why don't you post your code?

--

Nick Jackiw

...

From: amanda@mermaid.intercon.com (Amanda Walker)

Subject: Re: List Bummers Revisited (simple custom ldef with code)

USENET Macintosh Programmer's Guide

In article <cZVSJ9i00WB2Q63mcX@andrew.cmu.edu>, es2q+@andrew.cmu.edu (Erik Warren Selberg) writes:

> Is there any way to access
> variables in the main program from an LDEF, and if so, how??!

Well, they way I do it is to have a baby LDEF that, by default, just draws a string they way LDEF 0 does, but if the list's refCon is non-zero, treats it as a pointer to a function and calls it to do the drawing. All you have to do is make sure A5 is set correctly.

Here's the source for the LDEF in MPW C 3.0, taken in its entirety from known working code. It does contain "the cast from Hell," but hey :-).

```
-----
/*
   Simple Custom LDEF

   Amanda Walker, InterCon Corporation
   6 October 1988

   This is a very simple custom LDEF.  It acts just like LDEF 0 if the refCon
   field of the list is 0.  If it is non-zero, the LDEF treats it as the
   address of a routine to call to actually draw a list element.

*/

#include <Quickdraw.h>
#include <Lists.h>
#include <OSUtils.h>

#define nil ((void *) 0L)

pascal void STUBLDEF(lMessage, lSelect, lRect, lCell, lDataOffset, lDataLen,
                    lHandle)
short lMessage;
Boolean lSelect;
Rect *lRect;
Cell *lCell;
short lDataOffset, lDataLen;
ListHandle lHandle;
{
    char *p;
    long savedA5;

    /* Make sure A5 is valid so the draw code can use its globals. */
    savedA5 = SetCurrentA5();

    p = (char *) 0x938;    /* HiliteMode for color QD */
    lMessage--;            /* no initialization */
    if (!lMessage--) {     /* draw the cell */
        if ((*lHandle).refCon)
            (*((pascal void (*)(Rect *, Cell *, short, short, ListHandle))
              (*lHandle).refCon))(lRect, lCell, lDataOffset, lDataLen,
                                  lHandle);
        else {
            MoveTo(lRect->left + 5, lRect->top + 12);
            HLock((*lHandle).cells);
            DrawText((*lHandle).cells, lDataOffset, lDataLen);
            HUnlock((*lHandle).cells);
        }
    }
    if (lSelect) {
        *p = *p & 0x7f;
    }
}
```

```
        InvertRect(lRect);  
    }  
} else if (!lMessage--) {    /* toggle highlighting */  
    *p = *p & 0x7f;
```

USENET Macintosh Programmer's Guide

```
InvertRect (lRect);  
}  
SetA5 (savedA5);      /* no closing code */  
}
```

If you don't know how to build code resources with MPW C, take a look at the examples in the CExamples directory.

Hope this helps,

Amanda Walker InterCon Systems Corporation

•••

From: tim@hoptoad.uucp (Tim Maroney)

Subject: Re: List Bummers Revisited (and Debugging Help)

In article <cZVSJ9i00WB2Q63mcX@andrew.cmu.edu> es2q+@andrew.cmu.edu (Erik Warren Selberg) writes:

>ok... I finally fixed the clicking bug (by SetPorting every time I checked
>if the application was mine (& thus did something with it).

Great, send me a million dollars now.

>I'd like to put the data object I have (called TFile, an object with a bunch
>of neat stuff contained) into the list instead of using up 100 bytes a shot
>as I am now... I've tried putting it in using StripAddress, putting in the
>starting file (well, a pointer to the starting file) in the list's refCon and
>trying to access it from there, and nothing. Is there any way to access
>variables in the main program from an LDEF, and if so, how??! what am I doing
>wrong?

I don't know; once again, your report has been far too vague to allow any solid deductions or even founded speculations. "And nothing" is no more a report of a symptom that is "I don't feel so hot". You can store common data structures in the list's refCon or in the list's userHandle. These are then accessible by the LDEF through the list handle it is passed, and by the application through the list handle it is keeping track of. What's the problem? Does the refCon come out as zero or some other invalid value in the LDEF? Then you're not stashing the value correctly. The syntax would be "list^^.refCon = LONGINT(<whatever>);" if I remember my Pascal.

>also: is there any way to debug an LDEF (or any code resource) from LSP?
>this trial & error stuff is rather depressing!

Lots of ways. First, to debug on the Mac, you simply must learn a low level debugger, such as MacsBug or TMON. You can debug anything this way. David Oster just gave a description of the process. Briefly, place a Debugger instruction (0xa9ff) in your code at the point where you want to start watching the fun, and then step through. It helps to have a paper listing of your program so you can follow along better.

You can also track the general flow of execution in various other ways. If you're not sure a piece of code is getting executed, put a SysBeep in it. And so on.

If you simply can't live without a symbolic debugger, then you use a pass-through code resource. Your actual LDEF resource is six bytes, starting with the word for a JMP.L instruction with an absolute long operand (0x4ef9), and having the address of your definition procedure in the second and third words. You link the definition procedure (the LDEF code) with your application, and set up the six-byte LDEF resource with the address of the main LDEF routine before you create your list, making sure it's non-purgeable so it won't get refreshed from disk. Then you can put symbolic breakpoints in your LDEF to your heart's content. After it's debugged sufficiently, you can make it a real, separate LDEF again if you wish -- or you can leave it the way it is.

--

Tim Maroney, Mac Software Consultant, sun!hoptoad!tim, tim@toad.com

•••

USENET Macintosh Programmer's Guide

From: olson@bootsie.UUCP (Eric Olson)
Subject: Re: List definitions and fonts...

USENET Macintosh Programmer's Guide

In article <4715@helios.ee.lbl.gov> beard@ux1.lbl.gov (Patrick C Beard) writes:

>In article <36618@cornell.UUCP> wayner@cs.cornell.edu (Peter Wayner) writes:

>#I just built a scrolling list of names in my window

>#and it works fine. I would like to change the font

>#of the list, though, to be something other than

>#the application font. I can't seem to find the

>#correct way to do this.

>#

>

>What I do is bracket all of my calls to the list manager that do drawing

>with calls to TextFont and TextSize with the appropriate font and size.

>Then, I restore the old font and size to whatever is needed. This

>seems to work.

If you change the font after the list has been drawn, you may wish to set the indent field of the List Record. The LDEF sets this during the init message, but then doesn't expect the font to change. Something like:

```
GetFontInfo(&fontinfo);
(*macList)->indent.h = 4;
(*macList)->indent.v = fontinfo.ascent;
```

These are the values that Apple's LDEF 0 chooses at list init time.

Note that you only need to change this if the font in the list changes dynamically, not if you just change it before calling LNew.

-Eric

•••

From: ech@cbnewsk.att.com (ned.horvath)

Subject: Re: Cdevs and the List Manager

From article <13580@unix.SRI.COM>, by mxmora@unix.SRI.COM (Matt Mora):

> ...The whole idea of

> the update routine is to only draw what really needs to be updated. Not to

> invalidate the whole mess and redraw it.

>

> Does anybody know the real reason why the listmanager sometimes doesn't

> update correctly?

To add a bit more fuel to this thread: I've noticed that LUpdate does a ValidRect on the part of the window where the horizontal scrollbar is drawn -- even if the list has no horizontal scrollbar. I traced this one through the 6.0.5 PACK0 code: sure enough, it tests for the presence of a scrollbar, then (if there isn't one), branches around the draw routine to the ValidRect.

If LUpdate is only being called between BeginUpdate and EndUpdate, this doesn't matter. But if LUpdate is being called after resizing the list, this is annoying. The only workaround I've found is to explicitly call InvalRect after LUpdate. I have not checked to see if the same problem exists when there is no vertical scrollbar.

=Ned Horvath=

•••