Technical Note TB525

Dialog Manager Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic--questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the Macintosh Technical Q&A's web site.

[Sep 01 1993]

CloseDialog and item list disposal

When allocating memory for a dialog record manually instead of letting the Toolbox do it, New Inside Macintosh recommends calling `CloseDialog` instead of `DisposeDialog`. However, doing so causes a memory leak, because `GetNewDialog` copies `the DITL resource in memory. The DITL copy isn't released when calling CloseDialog`, and it isn't purgeable, even if the original DITL was purgeable. What's the official method of completely getting rid of a dialog whose storage you have allocated by hand?

`CloseDialog` was intended to mirror `NewDialog`; it allows you to close a dialog which you provided the storage for, including the item list. If you examine Inside Macintosh Volume I, page 414, you'll find that the item list is specifically not disposed of by `CloseDialog`, so it's acting as documented. It does this so it won't dispose of a dialog item list you might be planning on using again. If you do want to dispose of the item list, just do so after calling `CloseDialog`.

Back to top

Using an 'ictb' with a Monitors extension

Date Written: 8/27/92

Last reviewed: 11/24/92

How can I change the text size and font of a static text item in a Monitors control panel extension? The utility I use for creating the resources I need supposedly is doing it with an `'ictb'` resource that holds the information, but when I run the cdev the default font doesn't seem to have changed. I can do it with a useritem but I'd rather have the flexibility of a generic DITL item.

These are the issues to keep in mind when using an 'ictb' with a monitors extension:

1. Your extension has to contain a `'dctb'` (ID = -4040). This is needed in order to have the system create a color window for the dialog. When `GetNewDialog` is called, the system checks for a dialog color table of the same ID; if found, a color port is created and the `'ictb'` is looked for. When no `'dctb'` is present, the dialog is based in an old style port and not item color table is used (even if the `'ictb'` is present). The `'dctb'` can be limited to one entry, as in the following example:

```
resource 'dctb' (-4040) {
  {  /* array ColorSpec: 1 elements */
    /* [1] */
    wContentColor, 65535, 65535, 65535
  }
```

2. The `'ictb'` resource has to take into consideration the items from the normal "Options" dialog before modifying the items for the extension. Basically you need to leave the first ten items unchanged and then add the records that affect your own items. In the following sample I am using the `'icb'` to change items 1 and 8 in my extension's list, this means that in the `'ictb'` I am including the records that affect my items are 11 through 18 since the current implementation of monitors has 10 items of its own. The resource I am using is:

```
data 'ictb' (-4040) {
  "$0000 0000 0000 0000 0000 0000 0000 0000"            /* first ten two word   */
  "$0000 0000 0000 0000 0000 0000 0000 0000"            /* entries set to zero  */
  "$0000 0000 0000 0000 A00F 0058 0000 0000"            /* ..................... */
  "$0000 0000 0000 0000 0000 0000 0000 0000"            /* ..................... */
  "$0000 0000 A00F 0073 0000 0000 0000 0000"            /* ..................... */
  "$0000 0000 0000 0000 006C 0100 0009 7F7F"            /* .........l.......... */
  "$0000 0000 0007 F77F 0000 0010 0647 656E"            /* .............Gen.... */
  "$6576 6100 8704 0000 0AAA AA00 00AA AAAA"            /* eva.................. */
  "$AAAA AAAA AA00 0006 5379 6D62 6F6C"                 /* ..............Symbol */
```

Note that this 'ictb' will need rev'ing if Monitors ever changes and uses a different number of items in the Options dialog.

Check the Dialog Manager chapter of *Inside Macintosh* Volume V for more details on 'dctb's and 'ictb's. Also, there's an entry in the Q&A Tech Notes the Developers CD that goes into some detail regarding 'ictb's.

Back to top

9-point Geneva with a userItem proc requires TextFont & TextSize

Date Written: 11/1/90

Last reviewed: 8/1/92

Is there any way to stop the Macintosh Dialog Manager from playing with the txSize and txFace fields of a dialog's grafPort so that I can draw Geneva 9-point text from within a userItem proc?

Unfortunately, because the Dialog Manager forgets about your previous calls to TextFont and TextSize when you put up your dialog again, you will need to call TextFont and TextSize every time your userItem proc is called.

Back to top

Using Geneva 9 in Macintosh dialog text fields

Date Written: 11/7/90

Last reviewed: 6/14/93

In a Geneva 9 Macintosh dialog edit text item, the last visible token that extends beyond the end of the item's rectangle is not drawn when first displayed. Typing a character into the invisible part of the string results in the character before and all characters after the typed character being drawn. The problem always appears in the same edit text items but not in every edit text item. The only difference seems to be the width of item rectangles. What's wrong?

Here is some sample code that should show how to set up your dialog to better support Geneva 9 for your dialog's text fields. I have used it in my own code and everything seems to work OK.

```
myDialog := DialogPtr(NewPtr(SIZEOF(DialogRecord)));
if myDialog = nil then
  ExitToShell;
for i := 1 to 3 do
  begin
    if EventAvail(everyEvent, evt) then
  end;
myDialog := GetNewDialog(rDialog, Ptr(myDialog), DialogPtr(-1));
SetPort(myDialog);
dp := DialogPeek(myDialog);
TextFont(geneva);
TextSize(9);
dp^.textH^^.txFont := geneva;
dp^.textH^^.txSize := 9;
SetWTitle(myDialog, 'ADSP Thing');
  {do other dialog setup stuff here, like putting your text in your dItems.}
```

Back to top

DlgCut, Copy and Paste are OK for modeless dialog boxes

Date Written: 12/12/90

Last reviewed: 6/14/93

A modeless dialog box is the only window that ever comes up in my Macintosh application. A keydown event along with the cmd cut, copy, or paste modifier key calls the DlgCut, DlgCopy, or DlgPaste routines, passing the dialog box handle returned from a GetNewDialog call. When this dialog box contains an enabled edit text item, the cut works, but the application pastes garbage or sometimes bombs.

DlgCut, DlgCopy, and DlgPaste are fine for what you are doing, and the procedure you describe for your implementation sounds like it should work fine.

You do not need to set the port (as long as you're checking that the front window is a dialog, which doesn't affect you since you only have one window); the routines take their action based on the item list attached to the dialog pointer you pass. In fact, DlgCut, DlgCopy, and DlgPaste work on the dialog you passed even if the dialog is not frontmost, which is why you need to check.

Back to top

Color dialog boxes on Macintosh 68000 systems

Date Written: 12/14/90

Last reviewed: 8/1/92

Why does a color dialog box with multiple fonts using both the ictb and dctb resources show up as a single font on 68000 machines?

Color QuickDraw is not loaded (implemented) on 68000 machines, therefore, 'ictb's are ignored. However, you may still be able to use multiple fonts on your dialogs if you use a userItem and draw them yourself. When you create a userItem, you can set all the necessary attributes, draw the graphics, and just about anything else you want to do with a dialog without trying to compete with the dialog manager. Whenever you try to force attributes into a dialog, such as fonts, you don't always get the update messages. This can lead to sloppy looking dialogs. The dialog manager will let you do what you want with your userItem, then all you need to do is to set the font attributes back to what they were before.

Below is a sample of how to do a userItem. It is not necessarily complete, but it is good for reference to get you started.

```
/* create a userItem that covers the whole text area you want to draw
(it can overlap other dialog items, like buttons and so on) and install
your procedure to it like so */

GetDItem(tdial,itemNumber,&isitem,&theHandle,&rect);
SetDItem(tdial,itemNumber,userItem,(Handle)doMyDrawing,&rect);
/* theHandle and rect are not necessary for you to keep track of, they're
just needed to fill out the call. */

Pascal    void doMyDrawing(DialgoPtr dialWindow, short theItem)
{short    theSize;
short    theFont;
SetPort(dialWindow);
theSize=dialWindow->txSize;   /* save the current size and font */
theFont=dialWindow->txFont;
DrawMeasureText();            /* do your drawing */
TextFont(theFont);            /* restore stuff */
TextSize(theSize);
```

X-Ref:

Inside Macintosh Volume I, page 421 (Dialog Manager Chapter)

Back to top

Where to find WDEF for movable modal dialog boxes

Date Written: 12/17/90

Last reviewed: 8/1/92

The WDEF for movable modal dialog boxes is available on the latest *Developer CD Series* disc as follows:

Developer Essentials:Technical Docs:Human Interface:

Human Interface Goodies:Movable-Modal WDEF 1.0.1

and on AppleLink:

Developer Support:Developer Services:

Developer Technical Support:Developer Essentials:...

Back to top

Sample resources for setting font & size of Macintosh text items

Date Written: 8/30/91

Last reviewed: 8/1/92

How can I get a Geneva 9-pt default for static text and text edit items in my dialog?

The Dialog manager provides a mechanism for changing the font/size of static and edit text items via the `'ictb'` (item color table) This structure is documented in *Inside Macintosh* Volume V, pages V-279 to V-281. Basically you create an `'ictb'` that has entries for each item that you want to modify. The only unfortunate aspect of this whole process is that we do not have rez templates for `'ictb'`s . So, I am including the source code to some sample resources that might help you get started.

```
----------------------- rez source code follows -----------------------
/* dlog stuff */
resource 'DLOG' (128) {
    {80, 74, 290, 464},
    dBoxProc,
    visible,
    goAway,
    0x0,
    128,
    ""
};

resource 'DITL' (128) {
    {    /* array DITLarray: 8 elements */
        /* [1] */
        {185, 305, 205, 363},
        Button {
            enabled,
            "Done"
        },
        /* [2] */
        {46, 290, 64, 369},
        CheckBox {
            enabled,
            "Memory"
        },
        /* [3] */
        {97, 290, 115, 369},
        CheckBox {
            enabled,
            "Memory"
        },
        /* [4] */
        {148, 290, 166, 369},
        CheckBox {
            enabled,
            "Memory"
        },
        /* [5] */
        {41, 12, 70, 275},
        EditText {
            enabled,
            ""
        },
        /* [6] */
        {92, 12, 121, 275},
        EditText {
            enabled,
            ""
        },
        /* [7] */
        {143, 12, 172, 275},
        EditText {
            enabled,
            ""
        },
        /* [8] */
        {5, 6, 21, 269},
        StaticText {
            disabled,
            "Type into them to see whats up"
        }
    }
};

resource 'dctb' (128) {
    {    /* array ColorSpec: 5 elements */
        /* [1] */
        wContentColor, 65535, 65535, 52428,
        /* [2] */
        wFrameColor, 0, 0, 0,
        /* [3] */
        wTextColor, 0, 0, 0,
        /* [4] */
        wHiliteColor, 0, 0, 0,
```

```
        /* [5] */
        wTitleBarColor, 65535, 65535, 65535
    }
};

data 'ictb' (128) {
    $"0020 0020"           /*$00 the button item */
    $"0000 0000"           /*$04 Check box 1 */
    $"0000 0000"           /*$08 Check box 2 */
    $"0000 0000"           /*$0C Check box 3 */
    $"0000 0000"           /*$10 Edit Text 1 */
    $"000D 0040"           /*$14 Edit Text 2 just change the family, */
                           /*                    size, and text color */
    $"0000 0000"           /*$18 Edit Text 3 change family and size, */
                           /*                          using font name */
    $"8005 0054"           /*$1C Stat Text 1 */
/* Start of the ictb items here */
    /* Color table for the done button */
    $"0000 0000"           /*$20 ccSeed */
    $"0000 0002"           /* Reserved / size of color table */
    $"0000 0000 0000 FFFF" /* cFrameColor, 65535,65535,52428 */
    $"0001 FFFF FFFF CCCC" /* cBodyColor, 0, 0, 0 */
    $"0002 0000 0000 FFFF" /* cTextColor, 0, 0, 65535 */
    /* Edit Text 2 item text/color info... */
    $"0001"                /*$40 diFont application font */
    $"0000"                /*$42 diStyle plain */
    $"000A"                /*$44 diSize whatever... */
    $"FFFF 8000 0000"      /*$46 forecolor */
    $"FFFF FFFF CCCC"      /*$4C backColor */
    $"0000"                /*$52 diMode */
    /* Edit Text 3 item text/color info... */
    $"0068"                /*$54 diFont application font */
    $"0000"                /*$56 diStyle plain */
    $"000C"                /*$58 diSize whatever... */
    $"0000 0000 0000"      /*$5A forecolor */
    $"FFFF FFFF CCCC"      /*$60 backColor */
    $"0000"                /*$66 diMode */
/* Start of the font name table here */
    $"0743 6F75 7269 6572" /*$68 font name Courier */
```

Back to top

Movable modal dialog WDEF 0 procID

Date Written: 9/23/91

Last reviewed: 8/1/92

We're using movable modal dialogs under both Systems 6 and 7. The standard System 7 WDEF 0 uses a `procID` of 5. The Apple WDEF ID 128 designed for System 6 usage requires a procID of 2503 (ID 128 * 16 + 5). Can `Gestalt` be used to detect at runtime whether WDEF 0 supports a procID of 5, so we can change the in-memory `'DLOG'` resource's procID to 2503 if necessary?

Also, are we free to use the WDEF (from latest *Developer CD Series* disc) in our commercial applications, or do some restrictions apply?

Please, use the WDEF that DTS provides on the CD with a clear conscience. You are free to use it all you like.

Unfortunatly the `MovableModal` WDEF is one of the many features that you cannot use gestalt to explicitly test for. However, what you should do in this case is simply test for system 7.0 or later, and if you have that then the new WDEF is available (I know we told you guys not to test the system version but in this instance its ok.) BTW the presence of the new DITL manipulation calls does not signify the movable modal WDEF so the gestalt test you were doing is not valid.

Back to top

C version of AppendDITL now in Snippets folder

Date Written: 12/17/91

Last reviewed: 8/1/92

I would like to add my own items to the Printing Manager's dialogs. Do you have a C version of the function `AppendDITL` from the Macintosh Technical Note "How To Add Items to the Print Dialogs"?

The C version of this Technote code is in the Snippets folder of the latest *Developer CD Series* disc (under the name PDlog Expand). This code was written in Think C 4.0.4, so you may need to make some slight modifications if you're working with v 5.0.

Back to top

Disabling the System 7 Application menu

Date Written: 1/30/92

Last reviewed: 6/14/93

I'm handling the events in my own in dialogs. How do I go about disabling the Application menu in System 7.0?

To disable the System 7 Application menu, you need to call `ModalDialog` with a filter proc that returns a 1 for the `itemHit` on the first event. This should be done after you create the window with either `GetNewDialog` or `NewDialog`. This will set some private Process Manager flags and disable the Application menu until the window is dismissed.

You should not ever have to do this, because if you are using a window of type `dBoxProc`, then the System will disable the application menu for you. It may be confusing or upsetting to your users if you use a window of a style other than dBoxProc and do not let them switch out to other applications.

If you have some compelling reason to use a different window type and you cannot allow your users to switch out, then this code will disable the application menu:

```
pascal Boolean FakeMakeModalFilter(DialogPtr dlg, EventRecord *evt,
                                   short *itemHit)
{

   *itemHit = 1;
   return true;
}

WindowPtr MakeModalWindow(short resID)
/*
    This function takes the resource id of a DLOG resource.
    It will open the window and then use the ModalDialog trick to make the
    Process Manager disable the Application menu under System 7.
*/
{
   WindowPtr   wp;
   short       fakeItem;

   wp = GetNewDialog(resID,NIL,(WindowPtr)-1);
   ModalDialog(NIL,&fakeItem);

   return wp;
}

void main()
/*  Sample function - - InitToolBox not included. */
{
    WindowPtr wp;
    InitToolBox();    // standard witch chant initialization
    wp = MakeModalWindow(128);
    /* Now we can handle all the events and the user will not be able to
       switch out via the Application menu, until the window wp is
       dismissed.
    */
```

Back to top

Bug with Macintosh color alert windows

Date Written: 3/11/92

Last reviewed: 8/1/92

My alert needs to use a color window so that a 3D-button CDEF will put up color buttons. I created an `'actb'` resource using ResEdit which causes the alert to use a color window, but the border of the alert dialog is dimmed as though it's disabled.

What you have found is a bonafide bug in the system software. There is nothing that you can do about it at this point. Instead, you should use a modal dialog in place of the alert (maybe write your own alert procedure?). What's happening is that the window is created as invisible when you have an `'actb'`, and in order not to disturb other window highlighting, `ShowHide` is used to show it. Thus your alert truly is inactive at that point in time (good thing too since it looks unhighlighted...)

Back to top

How to gray out & restore static text item of 'DITL' in a 'DLOG'

Date Written: 5/3/89

Last reviewed: 8/1/92

How can I "gray out" and restore a static text item of a 'DITL' in a 'DLOG'?

To change the text from black to gray:

```
SetPort(...);    { grafPtr of the dialog's port }
GetDItem(...)    { the static text item }
PenPat(gray);
PenMode(patBic);
PaintRect(...);  { the item's rect }
```

To restore the text to black:

```
SetPort(...);    { grafPtr of the dialog's port }
GetIText(...);   { the text to restore }
```

X-Ref:

QuickDraw

Back to top

Alert dialog not updating

Date Written: 5/3/89

Last reviewed: 6/14/93

My Alert dialog doesn't completely update if the application was in the background or if a screen dimmer was activated.

"When an alert is removed, if it was overlapping the default button of a previous alert, that button's bold outline won't be redrawn." *(Inside Macintosh* Volume I, page 419)

Alerts are intended to call the user's attention to something that requires immediate action. It would be confusing for a window to be drawing over an alert. MultiFinder and screen dimmers are not entirely consistent with the alert interface: a screen dimmer will draw over the alert, and an application in the background may need to display an alert.

If an application in the background wants to show an alert, it should use the Notification Manager. Otherwise, when the application gets switched into the foreground the Dialog Manager will not update the alert as it does when it was first drawn. An application can work around this by using a modal dialog. Have a user item installed, as the "Dialog Manager" chapter describes. When the dialog gets an update event, it will call the user item to redraw.

If you find that alerts don't do what you want, consider either using a filter proc or not using the Dialog Manager at all, and simply create and manage the alert window and contents in your application.

Back to top

How to create an 'ictb' resource

Date Written: 4/25/89

Last reviewed: 8/1/92

I am trying to implement colored controls in a dialog but I can not find the template for 'ictb' resources in any of the interface files in MPW. Can you help?

Unfortunately, MPW rez can not handle the complicated task of compiling an 'ictb', so you have to create it by hand. The following is a sample 'ictb' resource with its accompanying 'DLOG' and 'DITL' (plus a free 'dctb'):

```
resource 'DLOG' (333) {
   {100, 100, 300, 400},
   dBoxProc,
   invisible,
   noGoAway,
   0x0,
   333,
   ""
};

resource 'DITL' (333) {
   {   /* array DITLarray: 3 elements */
   /* [1] */
   {82, 93, 102,168},
   Button {
   enabled,
   "OK"
   };
   /* [2] */
   {119,46,137,240},
editText {
enabled, "this dialog has color palette!"
   };
 /* [3] */
   {32,74,52,220},
editText {
enabled, "New Stuff"
}
   }
 };

resource 'dctb' (333,"CDlog Stuff") {
0x0, 0x0,
{
wContentColor,0xFFFF,0xffff,0xFFFF,
wFrameColor,0x0000,0x0000,0x0000,
wTextColor,0x0000,0x0000,0x0000,
}
};

DATA 'ictb' (333,"CDlog ictb") {
$"0000 0000" /* first item OK normal */
$"A00F 000C" /* edit 1: change family,face, size, fore, back,offset */
$"200D 0020" /* edit 2: change family,size,fore,back */
/* Text style record for second item, edit text 1 */
$"0034 0100 000C" /* font name at offset $34, bold, size 12 */
$"7F7F 0000 0000"/* fore color */
$"0000 7F7F 0000 0001"/* back color + mode */
/* Text style record for third item, edit text 2 */
$"01F9 0000 0012"/* font number,normal,size 18 */
$"6666 0000 3333"/* fore color */
$"FFFF 8E24 182F 0000"/* back color + mode */
$"0647 656E 6576 61"/* Geneva, for item  */
};

/* this is the way you see the ictb when using resedit
00000000 0000 0000 A00F 000C ........
00000008 200D 0020 0034 0100  .. .4..
00000010 000C 7F7F 0000 0000 ........
00000018 0000 7F7F 0000 0001 ........
00000020 01F9 0000 0012 6666 ......ff
00000028 0000 3333 FFFF 8E24 ..33...$
00000030 182F 0000 0647 656E ./...Gen
00000038 6576 61             eva
```

Back to top

Colorized controls and TextEdit fields

Date Written: 3/9/92

Last reviewed: 6/14/93

I want my TextEdit fields and the controls themselves have a background color of White, to make it obvious what the user can modify. I'd like to be able to tell the control that its `controlRect` should be painted white first, which would really make it stand out (or at least the actual check box should be filled with white and not the background color). I've scoured the documentation and have not figured out a way to do this. The same thing with TextEdit: Unless I save/set/restore the Background color to white in every visible call I make to TextEdit, it uses the window's background color for its background color which is what I'm trying to avoid. Any ideas?

The answer to your colorized dialog dilemma can be found in the description and definition of an 'ictb' resource. Basically 'ictb's are a little-used, little-known feature of the Dialog Manager, which allows you to specify an item color table for dialogs so that you can colorize various parts of a dialog. For text items you can set the fore and back colors, and for buttons and such you can set frame, body, text and thumb colors. These resources are described in detail in *Inside Macintosh* Volume V, pages 279-82. (Several diagrams are incorrectly labelled 'dctb' when they should be 'ictb's; however, the rest of the information is correct.) Using resources of this type will allow you to colorize to your heart's content.

Back to top

Where to find System 6 WDEF for movable modal dialogs

Date Written: 12/17/90

Last reviewed: 6/14/93

Where can I find the Macintosh WDEF for supporting movable modal dialogs in System 6?

For the WDEF information you are looking for, try AppleLink with the following path: Developer Support: Developer Services: Developer Technical Support: Developer Essentials: Technical Documentation: Human Interface: Human Interface Goodies: Movable-Modal WDEF 1.01.sit. It is also on the current Developer CD in the Developer Essentials folder. To locate it quickly on the Developer CD, use Pathfinder to search for Movable-Modal.

Back to top

Alert ParamText problem and workaround

Date Written: 2/8/91

Last reviewed: 6/14/93

What causes the Macintosh call to an Alert to hang if one of an Alert's ParamText items contains "^1"?

You've noticed a ParamText problem that's existed for quite a while. Unfortunately, we don't know of an easy workaround, other than prefiltering the strings you're passing to ParamText to remove all the "^"s.

A more complicated workaround would be to use a normal modal dialog instead of Alert. You could build the whole message yourself and completely replace the appropriate dialog item with it using SetIText. Of course, you'd have to simulate the default button yourself.

Back to top

CouldDialog, CouldAlert, FreeDialog, FreeAlert not implemented

Date Written: 5/22/92

Last reviewed: 6/14/93

Here's a tidbit I stumbled across in *Inside Macintosh* Volume VI, page 3-10: the four Dialog Manager procedures CouldDialog, CouldAlert, FreeDialog, and FreeAlert are no longer supported. I use CouldDialog, and I happened to notice that it didn't work right when I tested it under System 7, but I reported it as a bug. Now you tell us that it's not guaranteed to work in System 7. I can't recall a trap ever becoming suddenly unsupported like this. What's the story?

The system software engineers felt that CouldDialog, CouldAlert, FreeDialog, and FreeAlert didn't do much good under System 6, since the Could calls never completely guaranteed that all dialog items were loaded in. These calls also caused problems in the beta versions of System 7. Relatively little software uses those traps anymore; like many things in *Inside Macintosh* Volume I, they're relics of the days when Macintosh programmers had to deal with desk accessory and floppy disk support issues. So these calls were simply patched out. In the final System 7, the traps return without doing anything.

Back to top

Downloadables

Acrobat version of this Note (K). Download