TransSkel Programmer's Notes

8: Dialog Events Revisited

Who to blame: Paul DuBois, dubois@primate.wisc.edu
Note creation date: 01/05/94
Note revision: 1.03
Last revision date: 04/23/94
TransSkel release: 3.06

This Note discusses some limitations of the implementation of `SkelDlogFilter()` described in TransSkel Programmer's Note 2, and how TransSkel 3.06 addresses them.

01/17/94 — Corrected bug in `FileFilterWrapper()` function.
04/15/94 — Added description of `SkelDlogTracksCursor()`, which is new in TransSkel 3.12. Added discussion of role of `SkelDlogDefaultItem()`, `SkelDlogCancelItem()` and `SkelDlogTracksCursor()` in System 7-only applications.
04/23/94 — Added description of drag region command-click handling, which is new in TransSkel 3.13.

Applications typically present modal dialogs or alerts like this:

```
ModalDialog (filter, &item);

result = Alert (resourceNum, filter);
```

Here, `filter` is a function your application can provide to handle particular events during dialog processing, or `nil` if you provide no such function.

One problem that occurs during dialog processing is the "orphan" event, for instance an update or activate for windows that lie beneath the dialog. Since `ModalDialog()` normally has no way to know how to route such events within your application, they go unprocessed (they're "orphaned"), leading to a variety of problems. You can supply a filter function to handle these problems, but then you must always supply a filter and all your filters have to duplicate a certain amount of functionality. (See TN TB 37 and TPN 2.)

To deal with this problem, TransSkel provides a standard filter function that's accessed by calling `SkelDlogFilter()`. This routine gives your application a mechanism for routing orphan events through TransSkel's usual event dispatching machinery.

For instance, the calls above can be revised as follows to allow orphan events to be processed:

```
ModalDialog (SkelDlogFilter (filter, doReturn), &item);
SkelRmveDlogFilter ();

result = Alert (resourceNum, SkelDlogFilter (filter, doReturn));
SkelRmveDlogFilter ();
```

The standard filter returned by `SkelDlogFilter()` dispatches events that aren't for the dialog by sending them through TransSkel's event router. In addition, if `doReturn` is true, it maps Return/Enter onto a click in the default button.

`SkelDlogFilter()` works well and is easy to use, but experience with TransSkel releases 3.02 through 3.05 allows some deficiencies to be readily identified:

- `SkelDlogFilter()` doesn't allow you to map Return/Enter onto any item other than the item that's specified as the default in the dialog record. This is unhelpful if, for instance, you want these keys to map to different buttons depending on the information specified in the other dialog fields.

- It doesn't allow you to map Escape or Command-period onto a Cancel button. You can do that by providing dialog- and alert-specific filter functions, of course, but that can involve a lot of code duplication. This is such a standard aspect of the Macintosh user interface that an easier method is desirable.

- `SkelDlogFilter()` expects filter functions to be defined a certain way. When it was written, this was not a problem, because all dialog and alert filters has the following structure:

```
pascal Boolean
Filter (DialogPtr dlog, EventRecord *evt, Integer *item)
{
}
```

  System 7 introduced another type of dialog filter, used in conjunction with the Standard File calls `CustomGetFile()` and `CustomPutFile()`. Filters for these calls look like this:

```
pascal Boolean
Filter (DialogPtr dlog, EventRecord *evt, Integer *item, void *data)
{
}
```

  Here, `data` is a pointer to arbitrary data associated with the filter function. Unfortunately, since `SkelDlogFilter()` assumes a different argument structure, it cannot be used with such a filter.

The rest of this Note describes how TransSkel now tries to solve these problems.

Specifying Default and Cancel Dialog Items

If the second argument to `SkelDlogFilter()` is `true`, the item specified in the dialog record is treated as the default button. However, if you want to specify a default item explicitly, you can now do so with `SkelDlogDefaultItem()`. In addition, you can specify a cancel item with `SkelDlogCancelItem()`. Each of these functions takes a single argument indicating the relevant item number, and affects processing by the filter returned by the last call to `SkelDlogFilter()`.

Illustrated below are some common situations involving `ModalDialog()` and how the new calls can be used to handle them. It's assumed that `p` is a variable of type `ModalFilterProcPtr`.

Note that if you use either of the new calls, you can't use `SkelDlogFilter()` in the argument list to your `ModalDialog()` call. You have to save the return value of `SkelDlogFilter()` in a variable which you later pass to `ModalDialog()`.

- The simplest case occurs when the item specified in the dialog record is the default button and there's no cancel button. Here you just pass `true` to `SkelDlogFilter()`:

```
ModalDialog (SkelDlogFilter (filter, true), &item);
SkelRmveDlogFilter ();
```

- To specify a default button explicitly, do this:

```
p = SkelDlogFilter (filter, false);
SkelDlogDefaultItem (myDefaultItem);
ModalDialog (p, &item);
SkelRmveDlogFilter ();
```

- If you want to use the default item from the dialog record, but specify a cancel button explicitly, do this:

```
p = SkelDlogFilter (filter, true);
SkelDlogCancelItem (myCancelItem);
ModalDialog (p, &item);
SkelRmveDlogFilter ();
```

- To specify both buttons explicitly, do this:

```
p = SkelDlogFilter (filter, false);
SkelDlogDefaultItem (myDefaultItem);
SkelDlogCancelItem (myCancelItem);
ModalDialog (p, &item);
SkelRmveDlogFilter ();
```

- In the case where the cancel button is the default button, you can do one of of the following, depending on whether you want to use the default item specified in the dialog record, or to specify the default item explicitly.

```
ModalDialog (SkelDlogFilter (filter, true), &item);
SkelRmveDlogFilter ();

p = SkelDlogFilter (filter, false);
SkelDlogDefaultItem (myCancelItem);
ModalDialog (p, &item);
SkelRmveDlogFilter ();
```

  Note that if you call `SkelDlogCancelItem()` to specify the cancel button in this situation, then Return, Enter, Escape and Command-period all map to the same button.

`SkelDlogFilter()` and Alerts

Alert calls always assume the item specified in the dialog record is the default. Don't try to change the default for alerts by calling `SkelDlogDefaultItem()`. You can, however, usefully specify a cancel item for alerts. For example:

```
p = SkelDlogFilter (filter, true);
SkelDlogCancelItem (myCancelItem);
result = Alert (resourceNum, p);
SkelRmveDlogFilter ();
```

Handling Other Types of Filter Functions

As indicated earlier, the filter function used by the System 7 calls `CustomGetFile()` and `CustomPutFile()` has this structure:

```
pascal Boolean
Filter (DialogPtr dlog, EventRecord *evt, Integer *item, void *data)
{
}
```

Since this argument structure differs from that expected by `SkelDlogFilter()`, you can't use `SkelDlogFilter()`. A different function (described below) is required, which is ugly because it multiplies functions for dealing with different filter types, even though the problem to be solved is the same. However, since the `data` argument allows essentially anything the filter finds necessary to be attached to it, I hope that Apple's intent in adding the `data` argument is to provide the mechanism allowing cessation of inventing new filter types.

To handle filter functions with a `data` argument, use the new function `SkelDlogFilterYD()`. This function is similar to `SkelDlogFilter()` except that the type of the function return value and the first argument is `ModalFilterYDProcPtr` rather than `ModalFilterProcPtr`:

```
ModalFilterYDProcPtr
SkelDlogFilterYD (ModalFilterYDProcPtr filter, Boolean doReturn);
```

`SkelDlogFilterYD()` doesn't take any data argument because there's no need. You pass the data argument to your Standard File call, which in turn passes it to the standard filter returned by `SkelDlogFilterYD()`.

Here's an example showing how to use `SkelDlogFilterYD()` in conjunction with `CustomGetFile()`. The example defines a function `MyStandardGetFile()` that can be used as a replacement for `StandardGetFile()`. The only difference is that `MyStandardGetFile()` makes sure that orphan events are handled.

```
/*
* FileFilterWrapper() translates the file filter used in the call
* to CustomGetFile() into the format used by StandardGetFile().
* If no filter was supplied, it accepts all files for display.
*/

static FileFilterProcPtr   wrapperFileFilter;

static pascal Boolean
FileFilterWrapper (ParmBlkPtr PB, void *data)
{
       if (wrapperFileFilter != nil)
              return ((*wrapperFileFilter) (PB));     /* ignore data argument */
       return (false);
}

pascal void
MyStandardGetFile (FileFilterProcPtr fileFilter, short numTypes,
                   SFTypeList typeList, StandardFileReply *reply)
{
Point  where;

       /*
        * Set up.  Save pointer to fileFilter in global variable so
        * FileFilterWrapper() use it.  Set where to (-1, -1) so dialog
        * will be centered on screen.
        */
```

```
        wrapperFileFilter = fileFilter;  /* save pointer to filter in global */
        SetPt (&where, -1, -1);          /* center dialog on screen */

        /*
         * Call file dialog, mapping caller's arguments into CustomGetFile()
         * and supplying dialog filter.
         */

        CustomGetFile(FileFilterWrapper,              /* fileFilter wrapper */
                      numTypes,
                      typeList,
                      reply,
                      0,                              /* use default template */
                      where,                          /* center on screen */
                      nil,                            /* no dialog hook */
                      SkelDlogFilterYD (nil, false),  /* add filter here */
                      nil,                            /* no activeList */
                      nil,                            /* no activateProc */
                      nil);                           /* no data */

        /* remove dialog filter */

        SkelRmveDlogFilter ();
    }
```

Observe that to remove a dialog filter installed with `SkelDlogFilterYD()`, you use `SkelRmveDlogFilter()`, just as you do with `SkelDlogFilter()`.

`CustomGetFile()` does its own key mapping, so in the example above, TransSkel is not told to do any mapping itself (the example passes false to `SkelDlogFilterYD()` and doesn't call `SkelDlogDefaultItem()` or `SkelDlogCancelItem()`). This is also how you would deal with `CustomPutFile()`. So why the second argument to `SkelDlogFilterYD()` if it's always going to be `false`? Answer: I'm not sure it will always be the right thing to inhibit the TransSkel filter from doing key mapping. Apple may in the future use filters of this type in contexts other than Standard File dialogs where it may make sense for the application to specify key mapping. If so, if may be reasonable for that second argument to be `true`.

Changing the Cursor in Edit Text Items

If it's desirable to change the cursor to an I-beam when the cursor is in an editable text item, you can call `SkelDlogTracksCursor()` to tell the standard filter function to do this. For example:

```
p = SkelDlogFilter (filter, true);
SkelDlogTracksCursor (true);
ModalDialog (filter, &item);
SkelRmveDlogFilter ();
```

`SkelDlogTracksCursor()` is not used for alerts, since they normally don't contain any edit text items. It also doesn't appear to be necessary to ask for cursor tracking when you call `SkelDlogFilterYD()` in conjunction with a System 7 Standard File dialog.

TransSkel's Dialog Filter and System 7

The original problem that `SkelDlogFilter()` was written to solve (update events that are generated but not handled) still exists in System 7. However, System 7 includes three functions

that may be used instead of TransSkel's button-handling and cursor-tracking functions associated with `SkelDlogFilter()`. These are shown below:

TransSkel	**System 7**
`SkelDlogDefaultItem()`	`SetDialogDefaultItem()`
`SkelDlogCancelItem()`	`SetDialogCancelItem()`
`SkelDlogTracksCursor()`	`SetDialogTracksCursor()`

If your application runs *only* under System 7, then you can use the System 7 routines to achieve the same effect as the corresponding TransSkel routines. The System 7 routines are invoked in a slightly different manner than the TransSkel routines. See TN TB 37 for details.

The TransSkel routines may be used in any application because they are not system-specific. (The exception is that cursor tracking will not work on systems earlier than System 3.2 because the `FindDItem()` trap did not exist until then. In this case tracking is simply not attempted, but your application need take no special action.)

One difference between `SkelDlogDefaultItem()` and `SetDialogDefaultItem()` is that `SetDialogDefaultItem()` draws the bold item around the default button for you. `SkelDlogDefaultItem()` does not. You must define a user item and call `SkelSetDlogButtonOutliner()`. See TPN 10 for details.

Allowing Underlying Windows to be Dragged Around

Apple now recommends that you should allow an underlying window to be dragged around when a modal dialog is displayed when the user clicks in the drag region of the window while holding the command key down (MHIG-145). The TransSkel dialog filter now implements this by passing such events through to the TransSkel event router. Command-dragging does not change the plane of a window. I know of few applications that actually implement this interface guideline while a modal dialog is displayed, but if you use the dialog filter, TransSkel will handle it for you. Whether any user of your applications will ever realize that it can be done is another question!

References

Inside Macintosh, Volume VI. The Standard File Package
Macintosh Technical Note TB 37: Pending Update Perils
TransSkel Programmer's Note 2: Orphan Dialog Events
TransSkel Programmer's Note 10: Button Outlining
Macintosh Human Interface Guidelines