TransSkel Programmer's Notes

12: Modeless and Movable Modal Dialogs

Who to blame: Paul DuBois, dubois@primate.wisc.edu
Note creation date: 04/27/94
Note revision: 1.01
Last revision date: 06/01/94
TransSkel release: 3.13

This Note describes changes made in TransSkel release 3.13 to provide better support for modeless and movable modal dialogs.

The first section of this Note describes the state of support for dialog processing up to and including TransSkel release 3.12. The sections following describe changes made in subsequent releases to provide better dialog support.

Dialog Processing as of Release 3.12

`SkelDlogFilter()` and the related TransSkel routines `SkelDlogDefaultItem()`, `SkelDlogCancelItem()`, and `SkelDlogTracksCursor()` provide a simple way for applications to access certain event-filtering capabilities when presenting modal dialogs and alerts. These routines are discussed in detail in TransSkel Programmer's Notes 2 and 8. They do the following:

- Update and activate events for non-dialog window are sent into the standard TransSkel event router so they can be processed normally. This avoids the problem of "orphan" events during dialog processing, i.e., events that are simply discarded by `ModalDialog()` as irrelevant.

- The Return, Enter, Escape, and Command-period keys can be mapped onto clicks in the default or cancel buttons.

- The cursor can be tracked so it changes to an I-beam whenever it's inside an editable text item.

The situation for dialogs other than modal dialogs is less promising:

- Support for modeless dialogs is present but deficient. You can create a modeless dialog and register it with TransSkel by calling `SkelDialog()`. This gives you automatic window dragging, updating, activating and item selection notification. However, applications are subject to the following limitations, which cannot be worked around easily:

  — You're notified of item selections, but that tells you only that a click occurred in the item, not where. For instance, if you implement a scroll bar using a user item, you cannot tell what part of the bar mouse clicks occur in.

  — You cannot track mouse clicks, because notification occurs only after an item has been selected. For instance, if you display a scroll bar, you cannot properly track clicks in the thumb region.

— You cannot map key clicks onto button clicks.

— You do not receive notification that the dialog is coming active or going inactive. This would be useful for highlighting controls and text selections, adjusting menus, etc.

— You do not receive update event notification for the dialog, in case the dialog needs updating in ways for which `UpdtDialog()` is insufficient. (E.g., drawing a bold outline around the default button when you don't use a user item for the outline.)

— You do not receive notification for null events while the dialog is frontmost. This would be useful for tracking the cursor and changing it to an I-beam whenever it's inside an edit text item.

- There is no support at all for movable modal dialogs.

- Even for modal dialogs, certain defects remain:

  — Apple now recommends that the user be allowed to Command-click in the drag region of windows beneath a modal dialog and drag them around under the dialog to review their contents (Macintosh Human Interface Guidelines, p. 145). It appears that very few applications actually provide this capability yet, but it certainly can be useful. TransSkel applications could provide this capability by supplying dialog-specific event filters, but the standard TransSkel filter could provide it on a more uniform basis rather than requiring the code to be duplicated in multiple applications.

  — Window-specific idle-time functions freeze when a modal dialog comes up. This is okay if the function is supposed to run only when the window is frontmost, but it also happens for functions that are supposed to execute even when the window isn't in front. This is because null events aren't sent into the router. The same problem occurs with the application idle-time function.

The fix for the modal dialog deficiencies is relatively simple: check for null events and Command-clicks in window drag regions and send them into the standard router. This change has been incorporated in TransSkel 3.13.

Providing support for movable modal dialogs and better support for modeless dialogs is more involved and forms the subject of the rest of this Note.

Modeless Dialog Support

The modeless dialog support deficiencies listed earlier stem from use of the following model for processing events:

- Call `IsDialogEvent()` to determine whether or not the event is for a dialog.
- If it is, call `DialogSelect()` to process the event.
- If it isn't, send the event through the non-dialog event routing machinery.

This is a fairly typical way to process events, but suffers the shortcomings already noted. The deficiencies could be rectified by changing `SkelDialog()` to take another argument, for specifying a filter function to be interposed between dialog event identification and processing. Conceptually, incorporating the filter into event processing results in a model like this:

- Determine whether or not an event is a dialog event.
- If it is, determine which dialog the event is for.
- If the dialog has a filter function, pass the event to the filter.
- If the filter doesn't handle the event, pass it to `DialogSelect()`.

Unfortunately, the Toolbox provides little support for determining which event a dialog is for. `IsDialogEvent()` can be used to determine whether an event is for a dialog, but not which one. `DialogSelect()` can tell you that, but by the time it returns, it's already processed the event! Thus, the event router itself must figure out the proper dialog. It turns out that the steps needed to make this determination are similar to the steps the router already takes to classify non-dialog events. Thus, instead of calling `IsDialogEvent()` and duplicating those steps, the non-dialog event routing machinery was modified to determine as it classifies an event whether the event is for a dialog, and if so, which one. The events considered to be for a dialog are:

- When a dialog is frontmost, null events, key clicks, and mouse clicks in the content region belong to the dialog.
- Update or activate events that reference a dialog in the event message belong to that dialog.

In each of these cases, the dialog is checked to see whether it has a filter. If it does, the filter is given first chance at the event. If the filter doesn't handle the event, it's passed to `DialogSelect()`.

Filter writing is discussed in a later section of this Note.

Movable Modal Dialog Support

Handling movable modal dialogs is very similar to handling modeless dialogs, except that you can't switch to other windows belonging to the application. Given the changes already made to the event router to better handle modeless dialogs, only one additional change is needed to handle movable modal dialogs: it beeps if you click outside the dialog window. (You can still click in the menu bar to select menu items, and you can Command-click in the drag region of another window to drag it around in its current plane under the dialog.)

Movable modal dialogs are registered with `SkelDialog()` just like modeless dialogs, and filter functions for them are written the same way. If the dialog is a movable modal dialog, `SkelDialog()` assigns a `skelWPropMovableModal` property to the dialog instead of a `skelWPropModeless property`.

Modeless and Movable Modal Dialog Filter Writing

Filters for modeless and movable modal dialogs are defined the same way as for modal dialogs:

```
pascal Boolean
MyFilter (DialogPtr dlog, EventRecord *evt, short *item)
{
}
```

The filter returns `false` if the event should be further processed by the event router. It returns `true` if the filter handled the event and detects an item hit, and the `item` parameter should be set to the item number. (If the filter doesn't handle the event, but doesn't want it processed further, it should set `evt->what` to `nullEvent` and return `false`.)

The port is switched to the dialog when it's activated, so normally the filter shouldn't have to be concerned about setting the port itself. Updates can occur for for the dialog when it's not frontmost, but the port is set before the filter is called for update events (and restored afterward).

Below are some examples how to use the filter function for various types of events.

Null Events

If the dialog has edit text items, you can track the cursor during null events and change it to an I-beam when it's in an edit text item:

```
if (evt->what == nullEvent)
	SkelSetDlogCursor (dlog);
```

The filter should return `false` for null events so that the event router continues to process it. This is important because the event router does other kinds of housekeeping when null events occur.

Key Events

A typical thing to do with key events is to map certain keys, like Return and Enter, onto clicks in the dialog's buttons. TransSkel provides a call to make it easy to perform key-to-button mapping in filter functions. When you see a key event in your filter function, do the following to find out if the key maps to a button, and if so, which one:

```
if (evt->what == keyDown)
{
	if (SkelDlogMapKeyToButton (dlog, evt, item, defaultItem, cancelItem))
		/* item contains item number here */
}
```

The filter should return `true` if the key maps to a button.

Update Events

Filters can be useful for update events if the usual updating done by the Dialog Manager doesn't draw everything you want drawn. For instance, you can draw a bold outline around the default button without creating a user item for doing so:

```
if (evt->what == updateEvt)
	SkelDrawButtonOutline (SkelGetDlogCtl (dlog, defaultItem));
```

For update events, the call to the filter occurs between calls to `BeginUpdate()` and `EndUpdate()`.

The filter should return `false` for update events or the event router won't call the Dialog Manager to update the rest of the dialog.

Activate Events

The usual thing to do when you detect an activate event in a dialog filter is to change the state of the items in the dialog window. For instance, you can set the highlighting for controls and the current text selection, and, if you have a default button, draw an outline in black or gray. You might also want to adjust menus.

The filter should return `false` for activate events so the event router passes the activate to `DialogSelect()` to let that routine do whatever else it needs to do.

Although you cannot switch to another window within the application when a movable modal dialog is frontmost, you can switch to another application. This means you should be prepared to respond to deactivates (not just activates) if you use an event filter with a movable modal dialog.

Mouse Clicks

[this section under construction]

The first thing to do is convert the point to local coordinates.

Miscellaneous

Some tasks require checking every time the event filter is called. Suppose you have an edit text item and you want the default button to be inactive whenever the text item contains no text. It's not sufficient to check the contents of the text item on null events, because there can be a time window during which the button may be enabled even though the text has nothing in it. If the user deletes all the text by selecting it and hitting Delete, followed immediately by typing Return, there can be two sequential non-null events, and the button will be active during both of them. Thus you must check the text item on every event, and do it before classifying the event. When the key click occurs, the filter discovers the text item empty and disables the button. Then when it checks the key click, it finds the button disabled and properly ignores the click.

Caveat

If you've looked at the code for `SkelDlogFilter()`, you know that the standard filter extracts events that `ModalDialog()` will ignore and sends them into TransSkel's event router for processing. This is not legal for filters passed to `SkelDialog()` for use with modeless and movable modal dialogs. Such filters are called from *within* the event router, so they should never send events into the router.

Problems

This section describes some difficulties you may discover in processing dialogs and how to work around them.

Losing Activate Events

Dialog filters can be used to change the highlighting of dialog items upon receipt of activates or deactivates. For instance, if you create a modeless dialog, make it visible, and then resume event processing, the filter will receive an activate. However, if you create a modeless dialog and make it visible, and then immediately create and display another window in the plane in front of the dialog before resuming event processing, you'll have a problem. Putting the second window up in front of the first cancels the activate for the

first, so you may not see any deactivate for it, either.

One solution to this is to process activates and updates immediately after you show each window:

```
ShowWindow (dlog);
SkelDoEvents (activMask + updateMask);
ShowWindow (wind);
SkelDoEvents (activMask + updateMask);
```

This forces the events to be generated and processed for each window. If you don't want the window contents drawn until the last window is visible, don't process updates until all windows have been shown:

```
ShowWindow (dlog);
SkelDoEvents (activMask);
ShowWindow (wind);
SkelDoEvents (activMask + updateMask);
```

Idling in the Background

`SkelDialog()` doesn't take any argument for an idle-time function, unlike `SkelWindow()`. You can do idle-time processing (such as tracking the cursor) by checking for null events in the filter function, but those are passed to the filter only when the dialog is frontmost. The lack of dialog-specific idle-time functions is based on the presumption that dialogs aren't used for continuous monitoring or display purposes, but only for interaction with the user while the dialog is frontmost. If you need to continuously check something even when the dialog isn't in front, you must install an application idle-time function and handle the task there.

References

Macintosh Human Interface Guidelines
TransSkel Programmer's Note 2: Orphan Dialog Events
TransSkel Programmer's Note 8: Dialog Events Revisited