TransSkel 3.04 — Release Notes and Compatibility Issues

This document is an adjunct to the TransSkel Programmer's Manual. It discusses the capabilities of TransSkel which are new with release 3.0, and those changes which introduce incompatibilities with previous releases. If you've never programmed with TransSkel before, you don't need to read this.

The differences between releases 2.0 and 3.0 of TransSkel are myriad. If you have applications developed with versions of TransSkel earlier than 3.0, they will have to be fixed before they will compile. Furthermore, the differences between versions 2.0 and 3.0 are greater than between 1.0 and 2.0.

Documentation

The main programming manual was much revised. A series of short documents, the TransSkel Programmer's Notes, were instituted in order to discuss miscellaneous TransSkel programming issues in detail.

The Quick Reference Guide has been eliminated. Since TransSkel is now prototyped, the header file *TransSkel.h* serves essentially the same purpose as the Quick Reference Guide.

Source code organization and programming support

- The header files *TransSkel.h* and *Compiler.h* should be used. *Compiler.h* defines types used to encourage portability when porting TransSkel to other languages or C compilers. *TransSkel.h* contains constants, types, and functions defining the TransSkel programming interface.

- Source code is fully prototyped.

- `SkelApple()` has been moved into a separate source file to allow less code to be linked into applications that do not call that function. Other new miscellaneous routines follow this pattern, too.

Code size

- This release of TransSkel is larger. 2.01 compiles to about 3000 bytes of code, whereas 3.0 exceeds 5000 bytes. On the other hand, 2.01 was written on a Macintosh

512KE, and most machines today have much more memory, so this seems reasonable.

- On the plus side, since some of the routines have been split out from the core, if you don't use them, they don't get linked in to your application.

Name changes to existing functions

- `SkelEventMask()`, `SkelEventHook()`, `SkelBackground()` and `SkelDlogMask()` are now named `SkelSetEventMask()`, `SkelSetEventHook()`, etc., for consistency with their inverses `SkelGetEventMask()`, `SkelGetEventHook()`, etc. `SkelGrowBounds()` was similarly renamed `SkelSetGrowBounds()`, even though it has no inverse.

- `SkelMain()` and `SkelWhoa()` are now `SkelEventLoop()` and `SkelStopEventLoop()`, which are more indicative of what they're for.

- `SkelClobber()` is now `SkelCleanup()`.

- The private event-dispatching function `DoEvent()` is now the public function `SkelRouteEvent()`. It handles a single event. There are instances where the application may have obtained an event outside of the main event loop and wishes to pass it to TransSkel for processing. This is of particular importance in dialog filters. See TPN 2 for more information.

Changes to function return values, argument structure, or behavior

- Initially TransSkel was written to require no header file, so all functions were written to return no value or an integer value. Since many of the "Get" functions were used to get non-integer values, a pointer argument was used into which the result was stored. For consistency, "Get" functions that were used to retrieve integer values were written this way as well.

  Since THINK C now supports function prototyping, and since TransSkel has become more complex, a header file is now used. Since it's desirable to supply prototypes now anyway for type-checking, it no longer makes sense to write "Get" functions to return no value. Unless they are used to get more than a single value, these functions have been rewritten to return a value of the type desired.

  This makes some operations clearer. For example, to add key-up events to the event mask, one would have had to write this before:

```
int    mask;
SkelGetEventMask (&mask);
SkelEventMask (mask | keyUpMask);
```

  Now one can dispense with the temporary variable, and combine the two function calls:

```
SkelSetEventMask (SkelGetEventMask () | keyUpMask);
```

- All functions returning success/failure are declared `Boolean`.This includes `SkelMenu()`, `SkelWindow()`, and `SkelDialog()`.

- Functions which return no value have been changed from `int` to `void`. Also, application-supplied window and menu handler routines that return no value (such as are passed to `SkelMenu()`, `SkelWindow()`, `SkelDialog()`), which should be declared `void`.

- Some menu and window handler functions, have a different argument structure.

- Key-click handlers take an additional argument for key code, in addition to character code and modifiers word. Key code is also used to indicate key-up events (the high bit — 0x80 — is set).

- `SkelMenu()` now takes an argument indicating whether the menu is a submenu (hierarchical).

- `SkelApple()` still takes a `StringPtr` as its first argument, but the string may now specify multiple menu items. This allows, for instance, a "Help" item to be installed in addition to the standard "About MyApp…" item. As a result, the callback function passed as the second argument is now called with the item number of the selection for non-desk accessory selections from the Apple menu, so that it can tell which item to respond to. Formerly it took no argument, because it could assume it was called as a result of selection of the first Apple menu item. This means the function is isomorphic in calling structure to the select function passed to `SkelMenu()`.

  `SkelApple()` also now uses a menu ID of 128 instead of 1 for the Apple menu, in order to kick the number out of the range reserved for system resources. The demonstration programs were modified as necessary to reflect this change. The ID is available as the defined constant `skelAppleMenuID`, so applications that define menu IDs in the source can use something like this:

```
typedef enum { appleMenuID = skelAppleMenuID, fileMenuID, editMenuID };
```

  For further details on `SkelApple()` changes, see TPN 1.

- Null events are passed to event hooks now.

New functions and capabilities

- `SkelQuery()` — Allows querying of execution environment. In order to be compatible with various Macintoshes and system software versions, TransSkel has to make a number of tests to determine its execution environment. The results of these tests are available through the `SkelQuery()` call, analogous to `SysEnvirons()`. This may allow the application to avoid repeating the same tests, should it need to make its own compatibility determinations.

- `SkelSetMenuHook()/SkelGetMenuHook()` — Allows installation of a hook to be called when mouse-downs in the menu bar occur. This allows the application the option of enabling or disabling menu items at that time rather than after every action that could change the status of some menu item.

- `SkelGetModifiers()` — Returns state of modifier keys for last event processed by `SkelRouteEvent()`.

- `SkelGetWindowDevice()` — Returns device containing window and largest useable area on that device.

- `SkelSetZoom()/SkelGetZoom()` — Used to support application-specific window zooming behavior.

- `SkelActivate()/SkelClose()` — Used to allow applications to activate/deactivate or close windows when the handler routine may be unknown.

- `SkelWindowRegistered()` — Used to find out whether or not a window has been registered with TransSkel.

- Multitasking (MultiFinder) support.

- Hierarchical menu support.

- Apple Event support.

- Multiple-monitor awareness for window dragging, sizing and zooming

- Key-up event processing. There was formerly no easy way to get TransSkel to pass key-up events to the application. Setting the system event mask of using `SkelSetEventMask()` to include `keyUpMask` was ineffective as key-ups were ignored in the event dispatcher anyway. It was possible to modify the system event mask and install an event inspection hook to intercept key-ups, but that was ugly and required the hook to duplicate work the event dispatcher does, such as determining the window to which the event was to be routed.

- TransSkel now by default still ignores key-ups, but the event dispatcher will look for them and pass them to the front window. To receive them, you must modify the system event mask so the system won't throw them away, and tell TransSkel to process them with

```
SkelSetEventMask(SkelGetEventMask() | keyUpMask)
```

This call works no matter what the current set of events is, since it simply adds key-ups to that set.

When the event dispatch routine finds a key-up, it passes it to the key-click handler to the front window, after setting the high bit (bit 7) of the key code argument. The key-click handler can distinguish key-downs from key-ups by the value of this bit:

```
void Key (char c, unsigned char code, Integer modifiers)
{
	if (code & 0x80)
		/* it's a key up */
	else
		/* it's a key down */
}
```

If you modify the system event mask, don't forget to reset it the application exits! (TN 202, TPN 3)

Note
Applications that don't care about key-up events need not specifically test for key-up events in key-click handlers because the default system event mask ignores such events anyway.

Cosmetic changes

The menu bar doesn't flicker from multiple redraws when application exits.

Zoomed windows are erased before being zoomed.

`SkelCleanup()` hides any windows that remain visible from front to back, which is faster and looks better.

Porting Notes

You should turn on “prototypes required” in the Options menu so that you'll be sure to catch all function name changes and argument list changes to TransSkel routines.

Be sure to check your key-click handlers and Apple menu handlers, since their argument structure has changed.

When you see “function does not match prototype” error messages for TransSkel routines, the most common cause of this is a literal string argument that does not have (`StringPtr`) in front of it. You can avoid this by turning on the “"\p" is unsigned char []” option in the THINK C settings..

If you get a link error of “SkelApple: undefined function” remember that that function is now split off into a source file separate from *TransSkel.c*. If you include a TransSkel library in your project, *SkelApple.c* should be part of the library. If you include source files directly, both *TransSkel.c* and *SkelApple.c* should be included.