# TransSkel Programmer's Notes

## 11: The TransSkel 3.11 Interface

Who to blame: Paul DuBois, dubois@primate.wisc.edu
Note creation date: 02/23/94
Note revision: 1.00
Last revision date:
TransSkel release: 3.11

As of release 3.11, TransSkel public functions are compiled using Pascal binding conventions, rather than using C binding conventions. This Note describes how to port pre-3.11 TransSkel applications so they'll work with release 3.11. It also discusses the rationale for making the change.

TransSkel is written in C, and up until release 3.09, the TransSkel library was compiled using C function binding conventions. As of release 3.11, all public interface functions provided by TransSkel are compiled using Pascal binding conventions. (3.10 was a transitional release and is a special case described later). In addition, any callback functions passed to TransSkel by the host application are expected to use Pascal conventions. This allows TransSkel to be used not only from with C applications, but with Pascal applications as well.

## Porting Applications Written for Releases 3.09 and Earlier

Unfortunately, the way the C and Pascal conventions use the stack to handle function arguments are incompatible, so TransSkel applications written for releases up to 3.09 need to be modified to work with the 3.11 interface.

The porting process is straightforward. The compiler will automatically compile your code to use Pascal conventions for calls to TransSkel applications as long as you include *TransSkel.h* in source files that make TransSkel calls. Since you should be including this file anyway, that should not present a problem.

Callback functions your application passes to TransSkel must be compiled to use Pascal conventions. With one exception, this means only that you add the keyword `pascal` to your function type specification. For example, menu item selection functions used to be defined like this:

```
void
MyMenuSelect (short item)
{
      /* process selection */
}
```

Now they should be defined like this:

```
pascal void
MyMenuSelect (short item)
{
      /* process selection */
}
```

Similarly, the mouse click function for window handlers used to be defined like this:

```
void
Mouse (Point where, long when, short modifiers)
{
       /* do click processing */
}
```

Now it's defined like this:

```
pascal void
Mouse (Point where, long when, short modifiers)
{
       /* do click processing */
}
```

The exception is the key click function for window handlers. It used to be defined like this:

```
void
Key (char c, unsigned char code, short modifiers)
{
       /* do key processing */
}
```

It's not sufficient to add the word `pascal` to this definition. For prototyped functions (which TransSkel uses, since it's fully prototyped), THINK C passes character arguments in the *high* byte of a two-byte stack value. Thus the character and code values would appear in the wrong half of the parameter values as Pascal procedures would see them, since the value is in the *low* byte for Pascal `char` parameters. To avoid this problem, key handlers should be written like this now:

```
pascal void
Key (short c, short code, short modifiers)
{
       /* do key processing */
}
```

The character arguments are now `short`; however, this is consistent with the way the THINK C headers declare `char` arguments in Toolbox function prototypes, so you can pass `c` directly to Toolbox functions.

## Porting the Lazy Way

Just open your project document and update it. The compiler will stop and complain whenever it finds that you're passing a callback to a TransSkel function that's not defined properly. That way you know what you need to change.

## Porting C Applications Written for Release 3.10

3.10 was a transitional release that allowed two versions of TransSkel to be compiled, one for C function bindings and one for Pascal bindings. This was done by defining the symbol `skelBinding` as either nothing or `pascal` in *TransSkel.h*, so that the library could be compiled one way or the other. The reference manual for the 3.10 release provided some guidelines for writing application code so it would be compatible with either interface. For instance, instead of writing a menu selection function like this:

```
void
MyMenuSelect (short item)
{
      /* process selection */
}
```

You would write it like this:

```
skelBinding void
MyMenuSelect (short item)
{
      /* process selection */
}
```

Instead of writing a key handler like this:

```
void
Key (char c, unsigned char code, short modifiers)
{
      /* do key processing */
}
```

You'd write it like this:

```
#if skelPascalCompat
skelBinding void
Key (short c, short code, short modifiers)
#else
void
Key (char c, unsigned char code, short modifiers)
#end
{
      /* do key processing */
}
```

`skelBinding` is no longer used, so any code written to use it must be revised.

If you wrote a key handler the ugly way shown above, change it to look like this:

```
pascal void
Key (short c, short code, short modifiers)
{
      /* do key processing */
}
```

For all other callbacks, just change `skelBinding` to `pascal`.

Release 3.10 provided the Pascal-compatible library as the *TransSkelPas* library document. If you included that library in your project document, remove it and include *TransSkel* instead.

## Porting Pascal Applications Written for Release 3.10

Pascal project documents written for release 3.10 included the *TransSkelPas* library document and the *TransSkelPas.intf* interface file. Remove these from your project and include the *TransSkel* library document and the *TransSkel.intf* interface file.

Pascal source files referred to TransSkel in the `uses` statement like this:

```
        uses
                TransSkelPas;
```

Change the `uses` statement to this:

```
        uses
                TransSkel;
```

## Rationale for Changing the Interface

The primary reason for modifying TransSkel 3.11 to provide a Pascal-compatible interface was to allow it to be used from within Pascal applications without translating TransSkel itself into Pascal. The suggestion for this change was made by Lionel Cons, who noted that TransSkel release 2 had indeed been translated into Pascal, but was out of date with respect to the current C version. Instead of retranslating C into Pascal, why not declare all public and callback functions as `pascal` functions and provide a binary library that can be linked into Pascal applications?

The primary objection to this is that the entire "installed base" of existing C applications would be immediately rendered incompatible with a Pascal-compatible TransSkel. And since I write in C (I dislike Pascal), that would include all my own applications.

To avoid rendering C applications imcompatible with a TransSkel that used Pascal function bindings, I rewrote TransSkel so it could be compiled two ways. C applications could continue to use TransSkel in the normal way, and Pascal applications could use a version *TransSkelPas* that used Pascal bindings. This was release 3.10.

The attempt to accommodate both binding types was a mistake, as shown by the following scenario. Suppose you write a library of functions. You can make it usable from either C or Pascal by writing all the interface functions as `pascal` functions. However, if that library makes even a single TransSkel call, you'd need two versions of it: one compiled to expect C-binding TransSkel, for use within C applications, and one compiled to expect Pascal-binding TransSkel, for use within Pascal applications. This quickly escalates into an intractable mess.

I decided it'd be better to bite the bullet to convert TransSkel to use Pascal bindings only, and do the necessary one-time conversion of projects that use it. I do apologize for the inconvenience.