TransSkel Programmer's Notes

13: Universal Header and Power Macintosh Support

Who to blame: Paul DuBois, dubois@primate.wisc.edu
Note creation date: 11/18/94
Note revision: 1.00
Last revision date:
TransSkel release: 3.18

This Note describes the changes made to TransSkel release 3.18 to support use of the universal header files and generation of native mode PowerPC code on the Power Macintosh.

Beginning with release 3.18, TransSkel is written to be compatible with the universal header files. The source has been changed to use universal procedure pointers based on the `UniversalProcPtr` type defined in those headers. Use of universal procedure pointers allows conformance to the procedural interface expected when running on the PowerPC chip, so that native mode PPC code can be generated.

Interface Changes

TransSkel now uses universal procedure pointer (UPP) types where appropriate, i.e., when routine descriptors rather than function pointers might be necessary depending on the type of code being generated (68K vs. PowerPC). This change affects the interface in the following ways:

The `SkelInitParams` structure has changed slightly. It used to be:

```
struct SkelInitParams
{
      short               skelMoreMasters;
      GrowZoneProcPtr     skelGzProc;
      SkelResumeProcPtr   skelResumeProc;
      Size                skelStackAdjust;
};
```

The grow zone member is now a UPP, so the structure looks like this:

```
struct SkelInitParams
{
      short               skelMoreMasters;
```

```
        GrowZoneUPP         skelGzProc;
        SkelResumeProcPtr   skelResumeProc;
        Size                skelStackAdjust;
};
```

Use of the `SkelInitParams` structure in the PowerPC environment is discussed in TPN 5.

The following functions now have different prototypes because they require UPP's instead of function pointers:

Old:

```
	pascal ModalFilterProcPtr
	SkelDlogFilter (ModalFilterProcPtr filter, Boolean doReturn);
	pascal ModalFilterYDProcPtr
	SkelDlogFilterYD (ModalFilterYDProcPtr filter, Boolean doReturn);
	pascal short
	SkelAlert (short alrtResNum, ModalFilterProcPtr filter, short positionType);
	pascal void
	SkelSetDlogProc (DialogPtr d, short item, SkelDlogItemProcPtr proc);
	pascal SkelDlogItemProcPtr
	SkelGetDlogProc (DialogPtr d, short item);
```

New:

```
	pascal ModalFilterUPP
	SkelDlogFilter (ModalFilterU     10/09/93     11/18scal ModalFilterYDUPP
	SkelDlogFilterYD (ModalFilterYDUPP filter, Boolean doReturn);
	pascal short
	SkelAlert (short alrtResNum, ModalFilterUPP filter, short positionType);
	pascal void
	SkelSetDlogProc (DialogPtr d, short item, UserItemUPP proc);
	pascal UserItemUPP
	SkelGetDlogProc (DialogPtr d, short item);
```

For compiling 68K code, the impact of these changes is negligible since the new UPP types are equivalent to the old non-UPP types. For instance, `ModalFilterProcPtr` and `ModalFilterUPP` are the same in the 68K environment. For compiling in the PowerPC environment, you'll need to change your application since you must pass routine descriptors instead of function pointers for UPP parameters.

Examples of the way the function calls listed above are used for the PowerPC environment can be seen in the source for the Button, DialogSkel, Filter, and MultiSkel demonstration applications. Search the source files for the `skelPPC` symbol.

Header File Compatibility Problems

The universal headers create universal procedure pointer (UPP) types as routine descriptors for PowerPC code generation and as `ProcPtr` types for 68K code generation. Relying on UPP type availability is a problem for people that don't have or don't use the universal headers, because UPP types aren't defined anywhere in the old Apple headers. One way to deal with this would be to stipulate that TransSkel no longer supports compilation with the older Apple headers. I felt that was an unreasonable constraint (at least for now), so instead TransSkel defines compatibility types and macros if the universal headers are unavailable. This works as follows:

- *TransSkel.h* first determines whether or not the universal headers are being used. It defines the symbol `skelUnivHeaders` as 1 if universal headers (and thus `UniversalProcPtr`'s) are available, `0` otherwise. The value of `skelUnivHeaders` is determined by testing whether or not `USESROUTINEDESCRIPTORS` is defined, since that macro is defined in the universal headers but not in the old Apple headers.

- When `skelUnivHeaders` is 0, it’s assumed that the types and macros associated with UPP’s are unavailable and compatibility workarounds are defined to compensate. *TransSkel.h* `typedef`’s some of the UPP types needed in the TransSkel source code to the equivalent non-UPP types and defines macros that emulate UPP-manipulation macros:

```
# if !skelUnivHeaders

typedef      ProcPtr                    UniversalProcPtr;
typedef      GrowZoneProcPtr            GrowZoneUPP;
typedef      ModalFilterProcPtr         ModalFilterUPP;
typedef      ModalFilterYDProcPtr              ModalFilterYDUPP;
typedef      pascal void (*UserItemUPP) (DialogPtr d, short item);

# define     NewModalFilterProc(proc)   (ModalFilterUPP)(proc)
# define     NewModalFilterYDProc(proc) (ModalFilterYDUPP)(proc)

# define     DisposeRoutineDescriptor(upp)     /* as nothing */

# endif /* !skelUnivHeaders */
```

This is done primarily for UPP types needed for interface function arguments or return values.

If you need to test for universal headers in your own code, you can do so like this:

```
#if skelUnivHeaders
       /* universal headers are being used */
# else
       /* universal headers are not being used */
# endif
```

The symbol `skelUnivHeaders` is tested rather than `USESROUTINEDESCRIPTORS` for several reasons:

- I believe that testing `USESROUTINEDESCRIPTORS` is a valid way of testing whether or not the universal headers are in use, but if that turns out not to be true, I can fix all code that needs to know about universal headers by changing the way `skelUnivHeaders` gets its value. If `USESROUTINEDESCRIPTORS` were tested directly, it would be necessary to edit each such test.

- You can override the value of `skelUnivHeaders` if you like by setting it in your prefix code. This is not true for `USESROUTINEDESCRIPTORS`, which should be left alone. One use for this would be if you want to require compilation using the universal headers: you can cause the compiler to complain when they are not used by defining `skelUnivHeaders` as 1, since this will cause errors whenever UPP types are encountered in your source.

- Testing whether or not universal headers are used is a stopgap measure until they are truly used "universally." Right now THINK C can be used with universal headers or the old Apple headers which know nothing about UPP's. Eventually (say, one or two years hence) I may just assume the universal headers are used. At that point I'll just unconditionally define `skelUnivHeaders` as `1` in *TransSkel.h*.

PowerPC Code Generation

If you need to know whether you're generating PowerPC code, the macro `skelPPC` can be used.
`skelPPC` is 1 if compiling PowerPC code, 0 if compiling 68K code. A value of 1 also implies that the universal headers are available, since no non-universal header method exists for generating PowerPC code. (A value of 0 implies nothing, however.)

`skelPPC` is simply a shorthand. The usual way to test for PowerPC code generation is:

```
#if defined(powerc) || defined (__powerc)
#endif
```

But it's easier to write:

```
#if skelPPC
#endif
```

Here's an example that shows how to compile code conditionally for the PowerPC or 68K environments. It comes from *MSkelHelp.c* in the MultiSkel demonstration application. The example shows how to pass a control action procedure to `TrackControl()`, in this case a scroll bar tracking procedure.

The action procedure is declared according to following prototype:

```
static pascal void
TrackScroll (ControlHandle theScroll, short partCode);
```

In order to pass the action procedure to `TrackControl()`, a pointer to the procedure is stored in either a routine descriptor or a scalar variable as follows:

```
/*
* Set up a variable to point to the scroll tracking procedure.  For 68K code this
* is just a direct pointer to TrackScroll().  For PowerPC code it is a
* routine descriptor into which the address of TrackScroll() is stuffed.
*/

# if skelPPC        /* PowerPC code */

static RoutineDescriptor   trackDesc =
             BUILD_ROUTINE_DESCRIPTOR(uppControlActionProcInfo, TrackScroll);
static ControlActionUPP    trackProc = (ControlActionUPP) &trackDesc;

# else                     /* 68K code */

static ControlActionUPP    trackProc = TrackScroll;

# endif
```

The preceding code sets `trackProc` to the value appropriate for the type of code being generated. To use `trackProc`, just pass it to `TrackControl()`:

```
partCode = TrackControl (helpScroll, pt, trackProc);
```