TransSkel Programmer's Notes

5: SkelInit() and SkelInitParams

Who to blame: Paul DuBois, dubois@primate.wisc.edu
Note creation date: 10/09/93
Note revision: 1.03
Last revision date: 11/18/94
TransSkel release: 3.04

This Note describes the `SkelInitParams` structure used by `SkelInit()` as of TransSkel release 3.04.

05/02/94 — Revised Note to change `ResumeProcPtr` to `SkelResumeProcPtr`.
11/18/94 — Revised Note for TransSkel 3.18 universal header/PowerPC changes.

Purpose and History of SkelInit()

The first TransSkel function an application calls is usually `SkelInit()`, which initializes the various Macintosh managers and determines some of the characteristics of the machine on which the application is running.

In the earliest releases of TransSkel, `SkelInit()` took no arguments:

```
void SkelInit(void);
```

This was somewhat limiting. For instance, an application may wish to call `MoreMasters()` to preallocate master blocks of memory pointers, and it may wish to install a grow zone procedure for dealing with low-memory conditions. `SkelInit()` is the most logical place to do these things, so it was changed (in release 2.00) to take two arguments:

```
void SkelInit(Integer noMasters, GrowZoneProc gzProc);
```

Here `noMasters` is the number of times to call `MoreMasters()` and `gzProc` is the grow zone procedure. This change introduced an incompatibility with previous releases of TransSkel and applications had to be changed accordingly.

The new `SkelInit()` was more capable, but it left other limitations unaddressed. `SkelInit()` calls `InitDialogs()`, which can be passed a pointer to a resume procedure to be called if a fatal system error occurs. However, `SkelInit()` just passed `nil` to `InitDialogs()`, leaving the application no way to specify the procedure if desired. `SkelInit()` also provided no way to adjust the application stack size.

It's possible to rectify these problems by adding more arguments to `SkelInit()`, but that's another incompatible change. Furthermore, future changes might necessitate yet more arguments, which would require applications to change the way they call `SkelInit()` again.

The New Calling Sequence

If an incompatible change is going to be made, it might as well be such as to limit the impact of additional changes in the future. The approach chosen was to modify `SkelInit()` to take a single argument, a pointer to a structure specifying the initialization parameters. As of release 3.04, `SkelInit()` is declared like this:

```
void SkelInit (SkelInitParamsPtr p);
```

To make it easy for applications requiring no special setup, default parameter values are used if the application passes `nil`.

Passing a pointer to a structure allows the `SkelInit()` calling sequence to remain the same forever. The structure itself may change someday, but that will cause little problem if the calling program follows the rules described in the next section.

The initialization parameters structure is declared like this:

```
typedef struct SkelInitParams SkelInitParams, *SkelInitParamsPtr;

struct SkelInitParams
{
       Integer             skelMoreMasters;
       GrowZoneUPP         skelGzProc;
       SkelResumeProcPtr   skelResumeProc;
       Size                skelStackAdjust;
};
```

The structure members are used as follows:

- `skelMoreMasters`

  The number of times to call `MoreMasters()`. The default is six times.

- `skelGzProc`

  The grow zone procedure. The default is `nil`. For 68K code, `GrowZoneUPP` is equivalent to `GrowZoneProcPtr`. For PowerPC code, `GrowZoneUPP` is a routine descriptor; you should not use a function pointer directly.

- `skelResumeProc`

  The resume procedure to be passed to `InitDialogs()`. As of System 7, the resume procedure is obsolete, so a value of `nil` is recommended. (`nil` is also the default.) You can still specify a non-`nil` value for pre-System 7 applications.

  `SkelResumeProcPtr` is a `typedef` for functions defined like this:

  ```
  pascal void MyResume (void);
  ```

  `SkelResumeProcPtr` is used rather than `ResumeProcPtr` as a compatibility workaround (the latter has disappeared from recent Apple header files). This structure member, unlike the grow zone member, was not converted to a UPP type when the universal headers became available, because resume procedures are obsolete on all systems which require UPP's.

- `skelStackAdjust`

  The amount by which to adjust the default stack size. The default is 0.

Compatibility Guidelines

The new calling sequence is incompatible with releases of TransSkel prior to 3.04, and older applications need to change the `SkelInit()` call. However, this need be done only once, and future modifications to TransSkel should have little impact. The `SkelInit()` calling sequence will remain stable (one argument, a pointer to a structure), and it will continue to be true that an application wishing to use the default initialization parameters can pass `nil` to `SkelInit()`.

If your application uses initialization parameters other than the defaults, it can be written to be compatible with `SkelInit()` in the event of future changes to the `SkelInitParams` structure. The thing you should *not* do is declare and statically initialize a `SkelInitParams` structure, e.g., like this:

```
SkelInitParams initParams =
{
       10,
       MyGrowZone,
       nil,
       8096
};

SkelInit (&initParams);
```

The problem with this approach is that it may fail if the internal structure of the `SkelInitParams` type changes. For instance, if new members are added to the end of the structure, the C compiler will initialize them to 0, but that may not be the appropriate default.

The proper way to specify non-default values for initialization parameters is to use the function `SkelGetInitParams()`. You pass it a pointer to a `SkelInitParams` structure, which TransSkel fills in with the default values. Then you can change any fields for which the defaults are unsuitable and pass the pointer to `SkelInit()`:

```
SkelInitParams initParams;

SkelGetInitParams (&initParams); /* get default values */
initParams.skelMoreMasters = 10; /* change fields appropriately */
initParams.skelGzProc = MyGrowZone;
initParams.skelStackAdjust= 8096;
SkelInit (&initParams);
```

This method makes sure you get the values you want for any structure members you change explicitly, and if any new structure elements are added in the future, they'll get the default values.

The Grow Zone Procedure and PowerPC Code

If you are compiling PowerPC code, the grow zone procedure should be a routine

descriptor, not a function pointer. To write your code so it can be compiled for either the 68K or PowerPC environments, you might do this:

```
#if skelPPC         /* PowerPC code */
RoutineDescriptor gzDesc =
      BUILD_ROUTINE_DESCRIPTOR(uppGrowZoneProcInfo, MyGrowZone);
GrowZoneUPP       gzProc = (GrowZoneUPP) &gzDesc;
#else               /* 68K code */
GrowZoneUPP       gzProc = MyGrowZone;
#endif

SkelInitParams initParams;

SkelGetInitParams (&initParams);
initParams.skelGzProc = gzProc;
SkelInit (&initParams);
```

Summary

To use the default initialization parameters, call `SkelInit()` like this:

```
SkelInit (nil);
```

Otherwise, declare a structure of type `SkelInitParams`. Pass it to `SkelGetInitParams()`, change the fields appropriately afterward, then pass the structure to `SkelInit()`:

```
SkelInitParams initParams;

SkelGetInitParams (&initParams);
/* ...change fields here... */
SkelInit (&initParams);
```