9. Initializations

This topic describes operations that must be done before executing other FaceWare commands. These operations should only be done once (i.e., they should not be part of routines or loops that are executed more than once).

FaceWare Initialization

The command DoInit must be executed before any other call to FaceWare modules. DoInit searches for the LoadIt module and, if necessary, opens the program's temporary resource file (the name of which can be passed in uName). Parameter c indicates which of the core FaceWare modules will be used by the program:

c ≥ 0 = all core modules are in use
c = -1 = FaceIt is not in use
c = -2 = FaceIt & ViewIt are not in use
c = -3 = FaceIt, ViewIt, & DialIt are not in use
c = -4 = no core modules are in use

Passing c = 0, for example, indicates that FaceIt, ViewIt, and UtilIt will be used and causes these modules to be loaded and initialized. Passing c = -4, on the other hand, skips initialization of the core modules and indicates that the only modules in use will be independent utility-type modules.

Parameter b in DoInit can be used to request that b kilobytes of extra stack space (beyond the default stack size - see discussion below) be allocated by LoadIt. This also results in LoadIt calling MaxApplZone to expand the heap to its limit. b = -1 can be used to disable this functionality (if your program calls MaxApplZone and sets its own stack size), but most programmers pass b = 0 to get the default stack size.

Parameter a is used to set bit flags that are used by FaceIt. See the "Commands" topic in the FaceIt Guide for a further description of the use of this parameter. If not using FaceIt, pass a = 0.

The minimum program code shown in the "Minimum Code" topic passes a = b = c = 0 to DoInit. This indicates that the program will be using all of the core modules, that the stack size should be set to the default size, and that none of the special options supported by FaceIt will be used.

More About Stack Space

The option of using parameter b to reset stack space when calling DoInit is designed to help those programmers who have programs that declare large arrays or records within routines, or pass large variables "by value" to routines. Such "local" variables are allocated on the "stack" which is a block of memory of fixed size located above the program heap. If the arrays or records allocated on the stack are larger than the available stack space, then the stack will clobber memory in the heap, leading to a System crash.

Another important point to understand about the stack is that its growth is cumulative as one routine calls another. For example, suppose routine A calls B, and then B calls C. If routine A declares a 1000-element integer*4 array, B declares a 2000-element real*8 array, and C declares a 500-element integer*2 array, then the total stack memory used to call C would be at least: (1000)(4) + (2000)(8) + (500)(2) = 21000 bytes or about 20K.

One way of increasing the stack space is to pass b > 0 when calling DoInit. Other options may be provided by your development environment (in which case you should pass b = -1 to disable DoInit's actions). Another approach is to minimize stack use by making all large arrays or records global to the program (this would include Fortran COMMON blocks), or allocate large variables as dynamic blocks in the program heap.

The toolbox call "StackSpace" can be used to check stack space from within a routine at runtime, but a calculation like that shown above will usually be enough to tell you that a problem exists. The PeekCt control shipped with ViewIt can also be added to any ViewIt window to show the current stack and heap space. Also note that the default stack space on older Macs (8K) is much smaller than that used with newer Macs, so having sufficient stack space on a newer Mac does not guarantee that it will work on an older one.

Record Initialization

In cases where a (non-core) FaceWare module requires use of an additional, global record (defined in a

"Stor" file), this record must usually be initialized with the command DoPrep. DoPrep uses the values passed to it to set up the record's "header" (its first 16 bytes) so that it can be used with the corresponding FaceWare module:

```
FaceIt(nil,DoPrep,a,b,c,0);         Pascal
FaceIt(0L,DoPrep,a,b,c,0L);         /* C */
call FaceIt(0,DoPrep,a,b,c,0)     !Fortran
```

where a is the record's memory address, b is the module's baseID, and c is its versID. See demo programs that contain calls to DoPrep for examples of its use. Note that DoPrep is not necessary to access any of the core modules since the fRec record used by them is automatically initialized when DoInit is called.

These extra, global records required by modules are often referred to as "shared records" to denote the fact that they are global records that are shared between the module and the main program.