U9. Memory

UtilIt supports commands that can be used to check free memory, dynamically allocate blocks, remove modules from memory, and clean up memory when quitting from within environments such as HyperCard and Prograph.

FaceWare's approach to heap memory management makes use of a minimum "buffer" of free memory that all modules attempt to maintain. The size in bytes of this buffer is saved in fHeapBuff, which has a default value of 16384 bytes (16K). The value of fHeapBuff can be changed by the program at any time after DoInit is called, although 16K is sufficient for most programs. You should think of fHeapBuff as representing the size of a shared pool of heap memory used for disk-based resource swapping under low memory conditions (i.e. memory into which fonts and other purgeable resources can be swapped).

FaceWare modules attempt to maintain at least fHeapBuff bytes of free heap memory by calling ChkMem or NewBlk before each action that would consume significant amounts of free memory. Note that this scheme does not protect the main program from making its own mistakes, and so we recommend that it too make use of ChkMem or NewBlk (or an equivalent scheme) when attempting to allocate large blocks of memory.

ChkMem 171 a,b,c,uResult

Attempts to find a contiguous block of free memory at least a + fHeapBuff bytes in size. If necessary, ChkMem will, in succession, compact memory, purge memory, request memory from other modules, and move the clipboard to disk until a contiguous block equal to or larger than the specified size is found.

The extent of the operations performed when searching for free memory can be optionally restricted via parameter b:

0 = no restrictions
1 = don't remove purgeable blocks
2 = don't request memory from other modules
4 = don't move the clipboard to disk

Parameter c can be optionally used to designate a number of bytes to use in place of fHeapBuff (i.e., if c > 0, a + c becomes the target block size).

The size in bytes of the largest contiguous block that is safe to allocate (= largest block - fHeapBuff) is returned in uResult. Thus a program calling ChkMem can simply check to see whether uResult ≥ a upon return.

A good way to think of parameter a is to view it as a gauge of how much work you are asking ChkMem to do. Small values will be easy to satisfy, require few changes to the heap, and return the smallest values of uResult. Larger values of a can ultimately force all purgeable blocks from the heap, and return the largest possible value of uResult.

NewBlk 172 a,b,c,uResult

Performs a ChkMem and, if at least a free bytes can be safely allocated, creates a new relocatable block, moves it high in the heap, locks it down, and returns a handle to the block in uResult. If unsuccessful, uResult is set equal to zero. Use the toolbox call DisposHandle to later remove such a block from memory. Parameters b and c have the same meaning as described above for ChkMem.

For example, the following code allocates an 80,000 byte block and saves its handle in "myHandle" (Pascal source).

```
FaceIt(nil,NewBlk,80000,0,0,0);
if (uResult <> 0) then
  begin
    myHandle := Handle(uResult);
    [do something with the block]
    DisposHandle(myHandle);
  end
```

*where the address of the block would be given by myHandle^ [Pascal], *myHandle [C], or long(myHandle) [Fortran].*

PrgCmd 173 a

Attempts to unlock the FCMD and/or FACE resource(s) belonging to the module(s) designated by a. Unlocking such resources will make them purgeable so that the memory they occupy can be used for

other purposes.
a = scope of FCMD/FACE resources in use by the main program that are to be unlocked (made purgeable)
0 = all FCMD/FACE resources in System & Appl. Heaps
1 = all FCMD/FACE resources in System Heap
2 = all FCMD/FACE resources in Application Heap
<32768 = resource ID of a single FCMD/FACE resource
other = address of single shared record (single instance)
PrgCmd will fail to unlock an FCMD or FACE resource if a shared record associated with the resource exists which is not included in the scope of a, or if the resource's purge bit is not set. For example, if a main program requested that an FCMD be unlocked (made purgeable), then this would not be done if the FCMD was part of the System file and was, at the same time, being used by another program.
The most common use of PrgCmd is to unload as many FaceWare modules as possible before performing a program operation that makes use of the recovered heap memory:
FaceIt(nil,PrgCmd,2,0,0,0);
The next call to the "FaceIt" dispatching procedure will then reload any necessary FaceWare modules.
WARNING: If you unload all FaceWare modules, consume most of the heap space, and then make a "FaceIt" call, you won't have enough memory to reload FaceWare modules! Thus the best use of PrgCmd is to recover memory for temporary purposes so that it can be released again before reloading modules.
Finally, note the some modules include FACE and other resource types that are loaded by the module itself (vs. LoadIt), meaning that PrgCmd will have no effect on such resources. In these cases a module-specific command will often be provided to directly unload such resources.
MODULE DEVELOPERS: If using PrgCmd from within a FaceWare module, be careful not to unload any modules that were used to call the module that is executing PrgCmd, nor execute code that moves memory after calling PrgCmd to unload one's self.

DoUnld -63 [none]

Cleans up FaceWare's use of current application heap by (1) closing all open ViewIt windows, (2) *disposing of all blocks in the heap that were dynamically allocated by UtilIt when DoInit was first called, and (3) calling PrgCmd with a = 2 to unload all FaceWare modules.*
DoUnld is useful in high-level environments where programs can be exited without quitting to the Finder (such as when switching stacks within HyperCard, or rerunning programs within the Prograph interpreter).
WARNINGS: Do not call DoUnld from within programs that make use of the FaceIt module, and do not make any calls to the "FaceIt" dispatching procedure after calling DoUnld.