Using Temporary Memory

You can use temporary memory routines to determine how much memory is available for temporary allocation, to allocate blocks of memory for temporary use, to lock and unlock relocatable blocks of temporary memory, and to release memory previously allocated for temporary use.

As indicated in **About Temporary Memory**, you do not need to use the special temporary memory routines to lock, unlock, or release temporary memory blocks if your application is executing under system 7.0. Instead, you can employ the usual **Memory Manager** routines to accomplish those tasks. You can use temporary memory longer under system 7.0 than under system 6.0. Before taking advantage of these two new features, however, you should make sure that they are present. Methods for doing this are presented in **Determining Features of Temporary Memory**.

Allocating Temporary Memory

You can request a block of memory for temporary use by calling the **TempNewHandle** function. This function attempts to allocate a new relocatable block of the specified size for temporary use. For example, to request a block that is one-quarter megabyte in size, you might issue these commands:

```
mySize = 0x40000;    //one-quarter megabyte

myHandle = TempNewHandle(mySize, myErr);
```

If the routine succeeds, it returns a handle to the block of memory. The block of memory returned by a successful call to **TempNewHandle** is unlocked. If an error occurs and the routine fails, it returns a NIL handle. You should always check for NIL handles before using any temporary memory. If you detect a NIL handle, the second parameter (in this example, myErr) contains the result code from the function.

Instead of asking for a specific amount of memory and then checking the returned handle to see if you actually got it, you might prefer to determine beforehand how much temporary memory is available. There are two functions that return information on the amount of free memory available for allocation using the temporary memory routines. First, you can use the **TempFreeMem** function, as follows:

```
long memFree;  //amount of free temporary memory

memFree = TempFreeMem ();
```

The result is a long integer containing the amount of free memory, in bytes, available for temporary allocation. It usually is not possible to allocate a block of this same size because of fragmentation due to nonrelocatable or locked blocks. Consequently, you'll probably want to use the function **TempMaxMem** to determine what is the largest contiguous block of space available. To allocate that block, you can write:

```
mySize = TempMaxMem(myGrow);

myHandle = TempNewHandle(mySize, myErr);
```

TempMaxMem compacts the heap zone and returns the size in bytes of the largest contiguous free block available for temporary allocation. (**TempMaxMem** is therefore analogous to **MaxMem**; see the discussion of **MaxMem** in the **Memory Manager** for full details on **MaxMem**.) The myGrow parameter is a variable parameter of type Size; after the function is called, it always contains 0 because the temporary memory does not come from the application's heap. Even when you use **TempMaxMem** to size that available memory, you should check the handle returned to make sure that is it not NIL.

The temporary memory routines include the **TempTopMem** function, originally designed to help you determine how much memory is available in the current executing environment. **TempTopMem** returns a pointer to the top of the addressable RAM space. Note that the return value indicates the total amount of usable machine memory, not the amount of memory available to your application, so you should not use **TempTopMem** to calculate the size of your application's memory partition. Because you can determine the amount of available memory by using the **Gestalt** function (using the gestaltLogicalRAMSize selector), you should consider the **TempTopMem** function to be obsolete.

Locking Temporary Memory

Under system 7.0, you can lock a relocatable block of temporary memory by calling the **Memory Manager** procedure **HLock**, thereby preventing that block from being moved within the heap zone. In system 6.0, you can do this by using the **TempHLock** procedure.

Unlocking Temporary Memory

To unlock a specified relocatable block of temporary memory, you can use the **HUnlock** procedure. Once again, you can accomplish the same result in system 6.0 by calling the **TempHUnlock** procedure.

Releasing Temporary Memory

When you finish using a block of memory that you allocated using **TempNewHandle**, you can release it by calling the **DisposHandle** routine.

In system 6.0, you can free temporary memory blocks by calling the **TempDisposeHandle** procedure.