About Temporary Memory

To operate efficiently with other applications in the system 7.0 environment, your application should contain a 'SIZE' resource that specifies both a minimum and a preferred memory partition size. The actual partition size allocated to your application upon launch is set to the preferred size if that much contiguous memory is available, or to some smaller size if that much contiguous memory is not available. Note that once the application is launched, its partition cannot change in size, so the application's heap must always remain within some predetermined limit.

Rather than specify a preferred partition size that is large enough to contain the largest possible application heap, you should specify a smaller but adequate partition size. When you need more memory for temporary use, you can use a set of temporary memory allocation services provided by the Operating System. The memory allocated using these services is known as *temporary memory.*

By using the temporary memory allocation routines, your application can request some additional memory for occasional short-term needs. For example, the Finder uses these temporary memory routines to secure buffer space to be used during file copy operations. Any available memory (that is, memory currently unallocated to any application's partition) is dedicated to this purpose. The Finder releases this memory as soon as the copy is completed, thus making the memory available to other applications or to the Operating System for launching new applications.

Because the requested amount of memory might not be available, you should not rely on always getting the memory you need when you issue a temporary memory request. You should make sure that your applications still work even if no temporary memory is available when you request it. For example, if the Finder cannot allocate a large temporary copy buffer, it performs the copying using a reserved small copy buffer from within its own heap zone. While the copying might take longer, it is nonetheless performed.

In system 6.0, any memory you allocated using the temporary memory routines had to be released before your next call to **GetNextEvent** or **WaitNextEvent**. In addition, you had to perform all operations on that temporary memory by using specialized MultiFinder routines rather than the usual **Memory Manager** routines. In system 7.0, both of these restrictions have been relaxed (though not completely removed). The memory you allocate using the temporary memory routines can now be used for longer intervals and does not need to be released before the next call to **WaitNextEvent**. In system 7.0, you can think of temporary memory simply as an extension of your application's heap. You should still use temporary memory for as short a time as possible, and you must release that memory before your application terminates.

Because temporary memory is taken from RAM that is reserved for (but not yet used by) other applications, you might prevent the user from being able to launch other applications by using too much temporary memory or by holding temporary memory for long periods of time. You can hold temporary memory indefinitely, however, in certain circumstances. For example, if the temporary memory is used for open files and the user can free that memory simply by closing those files, it is safe to hold onto that memory as long as necessary. But you should make sure not to lock temporary memory across

event calls.

Temporary memory is tracked (or monitored) for each application, so you must use it only for code that is running on an application's behalf. Moreover, because the Operating System frees all temporary memory allocated to an application when it quits or crashes, you should not use temporary memory for procedures such as VBL tasks or **Time Manager** tasks that you want to continue running after the application quits. Similarly, it is wise not to use temporary memory for interprocess buffers (that is, a buffer whose address is passed to another application in a high-level event) because the originating application could crash, thereby causing the temporary memory to be released before (or even while) the receiving application uses that memory.

Another main difference between temporary memory routines under system 6.0 and system 7.0 is that although you must still allocate temporary memory by using special routines, you can release and otherwise manipulate it by using normal **Memory Manager** routines. Even though you do not know where the additional memory comes from, you are encouraged to operate on that memory in the same way that you manipulate memory allocated from the application or system heaps. In short, temporary memory *allocation* remains specialized, but operations on *existing* temporary blocks can be performed by using **Memory Manager** routines.

Under system 7.0, the following **Memory Manager** routines work even if the handle or pointer is allocated by one of the temporary memory routines:

DisposHandle

EmptyHandle

GetHandleSize

HandleZone

HClrRBit

HGetState

HLock

HNoPurge

HPurge

HSetRBit

HSetState

HUnlock

ReallocHandle

RecoverHandle

SetHandleSize

Prior to system 7.0, you need to use these seven routines on temporary

memory:

TempFreeMem

TempMaxMem

TempDisposeHandle

TempHLock

TempHUnlock

TempNewHandle

TempTopMem

> **Note:** In system 6.0, these routines have the following names: **MFFreeMem**, **MFMaxMem**, **MFTempDisposHandle**, **MFTempHLock**, **MFTempHUnlock**, **MFTempNewHandle**, **MFTopMem**. For compatibility, you can continue to use these names.

Note that **TempDisposeHandle**, **TempHLock**, **TempHUnlock**, and **TempTopMem** are now obsolete, although they still work (for the sake of compatibility).

> **Warning:** Although you can determine the zone from which the temporary memory is generated (using **HandleZone**), do not use this information to make new blocks or perform heap operations on your own.

Some of these routines rely on the current zone if the handle has been purged. They have been made to work correctly in this case, even if the current zone is the application or system zone and the handle is from a temporary block.

The **Gestalt** function includes several selectors that return information about the temporary memory routines. For more information, see the section, **Determining Features of Temporary Memory** for examples of using **Gestalt** in this manner.