Technical Note TN1063
Inside Macintosh: Processes: Time Manager Addenda

CONTENTS

Some Basic Time Manager Rules

Setting Up tmReserved

About tmWakeup

The Microseconds Alternative

Undeferred Time Manager Task

The Native Time Manager

Time Manager Errors

Summary

References

Change History

Downloadables

This technote discusses a number of Time Manager issues that are not covered in the Time Manager chapter of *Inside Macintosh: Processes* .

This Note is intended for all developers who want to do time measurement using the Time Manager routines.

Updated: [Sep 01 1996]

Some Basic Time Manager Rules

When programming with the Time Manager, it is important to observe the following rules:

- For each Time Manager task that you insert (using `InsTime` or `InsXTime`), you must remove the same Time Manager task (using `RmvTime`) once and only once.
- While your Time Manager task is inserted (that is, between `InsTime` and `RmvTime`), you can prime the task multiple times. For example, the sequence `InsTime`, `PrimeTime`, fire, `PrimeTime`, fire, `PrimeTime`, fire, `RmvTime` is perfectly legal. Some developers always insert, prime and remove their Time Manager tasks. While legal, this is unnecessarily inefficient.
- If you remove a Time Manager task (using `RmvTime`), you must insert it again (using `InsTime` or `InsXTime`) before priming it.
- If you have primed a Time Manager task, you must not prime it again until it fires. This is explicitly called out in Inside Macintosh: Processes, page 3-11. If you need to cancel a Time Manager task, simply remove it using `RmvTime`. If you need to reschedule a Time Manager task, remove it, then reinsert it, then prime it again.
- Don't use `InsXTime` unless you want drift free timing. A common mistake is to install a drift free Time Manager task (using `InsXTime`), prime it for 1 second, let it fire, and then, 5 minutes later, prime it again for another 1 second. The task fires immediately because it was installed as a drift free task. To avoid this behavior, simply

use the original `InsTime` call.

This rules are not new; they are all explicitly or implicitly described in *Inside Macintosh: Processes* . However, recent systems now enforce these rules more strictly. If you break these rules, you may see one of the following symptoms:

- The system crashes with a `dsVMDeferredFuncTableFull` (112) system error.
- The system freezes because low-memory is trashed (prior to System 7.5.5).
- The system bus errors because of a fatal page fault.
- `PrimeTime` returns `qErr` (-1) when you attempt to prime a task that isn't installed.
- Timer Manager tasks not firing at the right time.

Note:
"Timer.h" defines the various Time Managers routines as void functions, which prevents you from getting the error result from `PrimeTime`. See Time Manager Errors for instructions on how to get this error result.

Back to top

Setting Up tmReserved

On page 3-8 of *Inside Macintosh: Processes* , it clearly states that both `tmWakeUp` and `tmReserved` should be set to 0 (as shown in Listing 1) prior to the first call to `InsXTime` when using the extended Time Manager.

```
theTMTask.tmWakeUp = 0;
theTMTask.tmReserved = 0;
InsXTime((QElemPtr)&theTMTask);
PrimeTime((QElemPtr)&theTMTask, 2000);
```

Listing 1. Set both `tmWakeUp` and `tmReserved` to 0 before calling `InsXTime`.

If you do want to do some time measurement, then you have to call `RmvTime` to get the current value of `tmCount`, which leads later to a new call to `InsXTime`, and a call to `PrimeTime` with a 0 delay which has a special meaning in that case. Although it appears, after much reading, rather clear that you leave the current value of `tmWakeUp` untouched in the `TMTask` structure, you can't be sure what to do about the value of `tmReserved`.

The truth is that prior to October, 1992 (System Software 7.1), you didn't care, but it's more of a concern now, since Apple slightly modified the behavior of the Time Manager to deal with performance issues.

If you use code like that in Listing 2 and leave `tmReserved` untouched, then after 127 calls your extended time task is converted into a non-extended time task (for a good but can't-be-disclosed reason) which, being waked up with a 0 delay `PrimeTime` (which has no special meaning for a non-extended time task), will suddenly be called and called again -- more frequently than it should be.

```
// WARNING: Don't use this code!
RmvTime((QElemPtr)&theTMTask);
remaining = theTMTask.tmCount;
InsXTime((QElemPtr)&theTMTask);
PrimeTime((QElemPtr)&theTMTask, 0);
// WARNING: Don't use this code!
```

Listing 2. Typical code for measuring time using the Time Manager.

So, if you perform that kind of time measurement, change your code to that shown in Listing 3.

```
RmvTime((QElemPtr)&theTMTask);
remaining = theTMTask.tmCount;
theTMTask.tmReserved = 0;
InsXTime((QElemPtr)&theTMTask);
PrimeTime((QElemPtr)&theTMTask, 0);
```

Listing 3. Correct code for measuring time using the Time Manager.

Since the Time Manager, prior to System Software 7.1, doesn't care about `tmReserved`, then you can set `tmReserved` to 0 before each call to `InsXTime` without checking the system version. You still have, of course, to ensure that the Time Manager you're using is the extended one (the response to `gestaltTimeMgrVersion` is `gestaltExtendedTimeMgr` (3) or greater).

Back to top

About tmWakeUp

The following sentence, also on page 3-8 in *Inside Macintosh: Processes,* is incorrect: "The `tmWakeUp` field contains the time at which the Time Manager task specified by `tmAddr` was last executed (or 0 if it has not yet been executed)." It should say: "The `tmWakeUp` field contains the time at which the Time Manager task specified by `tmAddr` is scheduled to be executed (or 0 if it has not yet been primed)."

Note:
Since the format of that field is undocumented and used internally by the Time Manager, developers are strongly discouraged anyway from performing any kind of calculation or comparison on the value of this field, since that format could change in the future.

Back to top

The Microseconds Alternative

Another way to perform time measurement would be to use the `Microseconds` call, which is much easier to use and less likely to change in the future.

```
pascal void Microseconds(UnsignedWide *microseconds);
```

Listing 4. The prototype for `Microseconds`.

Important:
`Microseconds` has a reputation for being inconveniently slow. Many developers have run performance tests and noticed that `Microseconds` can take over 10 microseconds to execute. Prior to the advent of the native Time Manager, `Microseconds` had a number of performance problems:

- It was implemented in 68K code.
- It relied on a time base which is very slow to access on modern computers.
- The algorithm was somewhat convoluted.
- It was patched by various system components.

The native Time Manager has significantly improved the performance of `Microseconds`. However, for best performance, your application should use the `UpTime` routine. `UpTime` is layered directly on top of the PowerPC Time Base Register (TBR) and is both faster and more accurate than `Microseconds`.

`UpTime` is exported by `InterfaceLib` on all computers running Mac OS 8.5 and higher, and by `DriverServicesLib` on all PCI-based computers regardless of the system version.

Back to top

Undeferred Time Manager Tasks

This section describes an optimization that you might want to employ when using the Time Manager in the presence of virtual memory (VM). Most developers will not be interested in this; however, all users of the Time Manager should heed the following warning.

WARNING:
Because there is an *extremely remote* possibility that the memory you have allocated for your Time Manager task contains the special value listed below, if you want to ensure the behavior defined in **Inside Macintosh: Memory**, you should always clear the `qLink` field in the `TMTask` before installing it.

As described in **Inside Macintosh: Memory**, Time Manager tasks are automatically deferred by the Virtual Memory (VM) system to avoid fatal page faults. This was done for backward compatibility with existing applications that use the Time Manager, but it can seriously increase the latency between when the timer expires and when your Time Manager task executes.

For more information about interactions between the Time Manager and VM, see Technote 1094 Virtual Memory Application Compatibility.

For example, if you schedule a Time Manager task to execute at time X and, at time (X - delta) some process takes a page fault, your Time Manager task will not be called until time (X + Y - delta), where Y is the time required to field a page fault. If the page fault causes the hard disk to seek, Y could be as great as the hard disk's average seek time, approximately 10 ms. If you are trying to use the Time Manager to measure time in *microseconds* , this could be a problem.

There is a way you can install Time Manager tasks so the callback is not deferred by VM; however, before using this technique, you should be aware of its dangers. Because VM does not defer these special Time Manager tasks, it is possible for them to fire when paging is not safe. To avoid fatal page faults, you must ensure:

- The `TMTask` record is held for the entire time the Time Manager task is installed (see Listing 5).
- The code for the timer task and any data it references is held. If the code for your timer task is stored in a code resource, you can use the snippet from Listing 6 to make sure it is held. If your timer task code is not in a code resource, it's very difficult to ensure that it and its data are held.
- You timer task code only calls system routines that are guaranteed to meet the above requirement -- this typically means only that routines that are known to be interrupt-safe.

```
HoldMemory(&theTask, sizeof(TMTask));
```

Listing 5. Holding the Time Managertask itself.

```
// Ensure the code doesn't move in logical memory
HLock(ttaskCodeHandle);
// Ensure the code is held in physical memory and cannot be paged to disk
HoldMemory(*ttaskCodeHandle, GetResourceSizeOnDisk(ttaskCodeHandle));
```

Listing 6. A source code listing showing some code that helps to explain my idea....

WARNING:
If you fail to meet these requirements, you will cause a fatal page fault and crash the system.

If you call `InsTime` or `InsXTime` with the `qLink` field set to $65616461, the VM patch on the Time Manager will recognize your special requirements and execute your timer task as soon as it fires, regardless of whether paging is safe or not.

Back to top

The Native Time Manager

In late 1999, Apple introduced a new implementation of the Time Manager, known as the native Time Manager. The native Time Manager has the following features:

- It publishes the same programming interface as the extended Time Manager.
- It is implemented as PowerPC native code.
- It uses the PowerPC Time Base Register (TBR) and Decrementer register (DEC) as its time source. Prior to the introduction of the native Time Manager, all timing was done using the timer registers in the VIA (6522 Versatile Interface Adapter) or, more accurately, the VIA cell in the system's I/O ASIC. This change yields both more accurate timing and faster execution time.
- The `Microseconds` call is also implemented in terms of TBR, so the Time Manager and `Microseconds` are synchronized.

The native Time Manager is available on all ROM-in-RAM CPUs running Mac OS 9 or later, and on CPUs with the Uni-N ASIC running Mac OS 8.6. The best way to detect the presence of the native Time Manager is to call `Gestalt` with the `gestaltTimeMgrVersion` selector and look for the result `gestaltNativeTimeMgr`.

Note:
The definition of `gestaltNativeTimeMgr`, shown below, is missing from Universal Interfaces 3.3.2 and below[2474617].

```
enum {
    gestaltNativeTimeMgr = 4  /* native Time Manager is present */
};
```

Listing 7. "Gestalt.h" additions for the native Time Manager.

Developer Consequences

The native Time Manager publishes exactly the same programming interface as the extended Time Manager, so the developer consequences of the new implementation should be minimal. The following sections describe the gotchas we have seen so far.

Testing for Time Manager Features

When testing whether a particular Time Manager feature is available, always compare the result of `gestaltTimeMgrVersion` with the "greater than" operator rather than the "equals" operator. For example, the code in Listing 8 tests whether the features of the extended Time Manager are available. If you test for equality, your code will do the wrong thing when the native Time Manager is available.

```
static Boolean HaveExtendedTimeManager(void)
{
    UInt32 response;

    return (Gestalt(gestaltTimeMgrVersion,
                    (SInt32 *) &response) == noErr)
            && (response >= gestaltExtendedTimeMgr);
}
```

Listing 8. Detecting the extended Time Manager.

Error Results

The native Time Manager can return an error in cases where previous versions of the Time Manager never did. It is important that you check and handle error results from calls to the Time Manager. See Time Manager Errors for details.

Back to top

Time Manager Errors

Since its initial release, the Time Manager's routines have always been defined to return an error result in register D0 (see *Inside Macintosh IV* , page 300). However, the high-level glue for calling these routines has never exposed these error results. This is particularly problematic for the native Time Manager, which can return an error in cases where previous versions of the Time Manager never did.

Apple has addressed this problem by defining new high-level interfaces for the Time Manager. The prototypes for the new routines are shown below.

```
extern pascal OSErr InstallTimeTask (QElemPtr tmTaskPtr);
extern pascal OSErr InstallXTimeTask(QElemPtr tmTaskPtr);
extern pascal OSErr PrimeTimeTask   (QElemPtr tmTaskPtr, long count);
extern pascal OSErr RemoveTimeTask  (QElemPtr tmTaskPtr);
```

Listing 9. New Time Manager high-level glue.

These routines are identical to the original Time Manager routines except that they return an error result. The routines were first exported by in Universal Interfaces 3.3, however they were only callable from 68K code. Calling these routines from PowerPC code required two extra steps.

- Mac OS 9.0.4 exports these routines from InterfaceLib.
- A future version (post-UI 3.3.2) of Universal Interfaces will include an InterfaceLib stub library that includes these routines.

You can simulate these routines for PowerPC code running on earlier systems by writing your own CFM glue. See DTS Technote 1127 In Search of Missing Links for the gory details. However, an easier solution is to use the glue provided by the MoreInterfaceLib module of the DTS sample code library MoreIsBetter.

Back to top

Summary

The following points explain what you should and should not do in working with the Time Manager.

- Always follow the basic rules.
- Always set `tmReserved` to 0 before calling `InsXTime`.
- Set `tmWakeUp` to 0 before the first call to `InsXTime`, never look at it or modify it (except to set it to 0 in some cases, no other value is acceptable) afterwards.
- `tmCount` is only valid after a call to `RmvTime`.
- Always clear `qLink` before calling `InsXTime`.
- `Microseconds` is good alternate way to measure elapsed time.
- Always check Time Manager error results.

References

Inside Macintosh: Processes , Chapter 3, The Time Manager

Denis G. Pelli's Web page

DTS Technote 1127 In Search of Missing Links

DTS Technote 1194 Mac OS 9.0.4

Back to top

Change History

September 1996	Originally written to elaborate on `tmReserved` and `tmWakeUp`.
March 1999	Updated to include information about undeferred Time Manager tasks.
September 1999	Updated to reiterate some basic Time Manager rules. Also made some minor clarifications and cosmetic changes.
July 2000	Updated to describe the native Time Manager and Time Manager errors.

Back to top

Downloadables

Acrobat version of this Note (K). Download

Back to top