Time Manager Versions

The **Time Manager** included in system software version 7.0 is the third version released. The three versions are all upwardly compatible-that is, each succeeding **Time Manager** version is a functional superset of the previous one, and code written for a version of the **Time Manager** executes on any later version. The reverse, however, is not true, and code written for the extended **Time Manager** may not execute properly on either the original or revised version. As a result, it is sometimes important to know which **Time Manager** version is available on a specific machine. You can use the **Gestalt** function to determine which version is present.

The Original Time Manager

The **Time Manager** was first introduced with the Macintosh Plus ROMs (which are also contained in the Macintosh 512K enhanced) and was intended for use internally by the Operating System. Volume IV of *Inside Macintosh* documented the routines in the original Time Manager, and thereafter some applications used them to schedule tasks to be executed at later times. The original Time Manager allows delays as small as 1 millisecond, resulting in a maximum range of about 24 days.

To schedule a task for later execution, you place an entry into the **Time Manager** queue and then activate it. All **Time Manager** routines manipulate elements of the **Time Manager** queue, which are stored in a **Time Manager** task record. The task record for the original **Time Manager** looks like this:

```
struct TMTask  // original and revised Time Manager task record
{
QElemPtr      qLink   ;         // next queue entry
short         qType   ;         // queue type
ProcPtr       tmAddr// pointer to task
Long          tmCount           // reserved
};
```

Of the four fields in this record, your application needs to supply only the tmAddr field, which contains a pointer to the routine that is to be executed at some time in the future. The remaining fields are used internally by the **Time Manager** or are reserved by Apple. Your application should set these remaining fields to 0 when you set up a task record.

The original Time Manager includes three routines:

- The **InsTime** procedure installs a task record into the **Time Manager** queue.
- The **PrimeTime** procedure schedules a previously queued task record for future execution.
- The **RmvTime** procedure removes a task record from the **Time Manager** queue.

Note that installing a request into the **Time Manager** queue (using the **InsTime** procedure) does not by itself schedule the specified routine for future execution. After you queue a request, you still need to activate (or

THINK Reference © 1991-1992 Symantec Corporation

prime) the request by specifying the desired delay until execution (using the **PrimeTime** procedure). Note also that the task record is not automatically removed from the **Time Manager** queue after the routine executes. As a result, the task may be reactivated when you subsequently call **PrimeTime**; you do not have to reinstall the task record. To remove a task record from the queue, you must call the **RmvTime** procedure. **RmvTime** removes a task record from the **Time Manager** queue whether or not that task was ever activated and whether or not its specified time delay has expired.

The Revised Time Manager

System software version 6.0.3 contains a revised version of the **Time Manager**. This version provides better time resolution and the ability to perform measurements of elapsed time with much greater accuracy. You can represent time delays in the revised **Time Manager** as microseconds as well as milliseconds, with a maximum resolution of 20 microseconds. The external programming interface did not change from the original to the revised **Time Manager**, although the revised version provides a means to distinguish microsecond delays from millisecond delays. The revised **Time Manager** interprets negative time values (which were not formerly allowed) as negated microseconds. For example, a value of -50 is interpreted as a delay of 50 microseconds. Positive time values continue to represent milliseconds. When you specify delays as microseconds, the maximum delay is about 35 minutes. When you specify delays as milliseconds in the revised **Time Manager**, the maximum delay is about 1 day. The delay specified to **PrimeTime** is converted to an internal form, so it makes no difference which unit you use if the delay falls within the ranges of both.

The revised **Time Manager** provides additional **Time Manager** features. The principal change concerns the tmCount field of the **Time Manager** task record (previously reserved for use by Apple). When you remove an active task from the revised **Time Manager** queue, any time remaining until the scheduled execution time is returned in the tmCount field. This change allows you to use the **Time Manager** to compute elapsed times (as explained in Computing Elapsed Time). In addition, the high-order bit of the qType field of the task record is now a flag to indicate whether the task timer is active. The **InsTime** procedure initially clears this bit, **PrimeTime** sets it, and it is cleared when the time expires or when your application calls **RmvTime**.

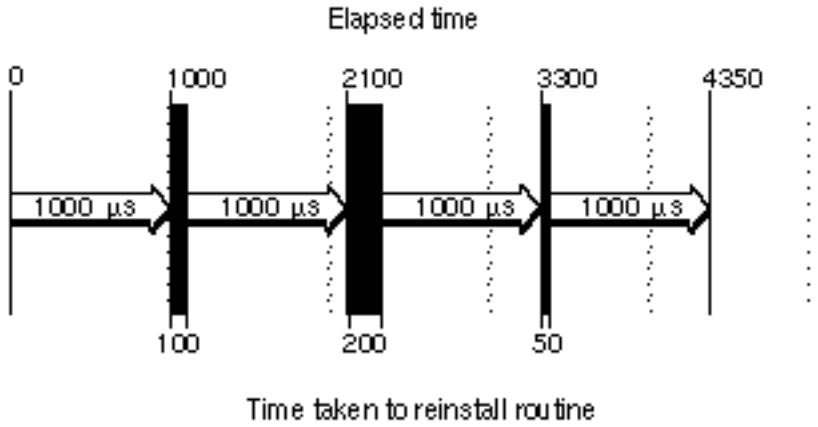
Although the revised **Time Manager** supports delay times specified in microseconds, you should use this feature primarily for the more accurate measurement of elapsed times. Applications that specify very small delay times in order to execute a routine at a high frequency may use a considerable amount of processor time. The amount of processor time consumed by such timing services varies, depending largely on the performance of the CPU. With low-performance CPUs, little or no time may be left for other processing on the system (for instance, moving the mouse or running the application).

The Extended Time Manager

The extended **Time Manager** (available with System 7.0) contains all the features of earlier Time Managers, with several extensions intended primarily to provide drift-free, fixed-frequency timing services. These services ensure that a routine executes promptly after a specified delay and are important for sound and multimedia applications that require precise timing and real-time synchronization among different events.

THINK Reference © 1991-1992 Symantec Corporation

In the original and revised Time Managers, the delay time passed to **PrimeTime** indicates a delay that is relative to the current time (that is, the time at which you execute **PrimeTime**). This presents problems when you need a fixed-frequency timing service and attempt to implement it by having the task reissue a **PrimeTime** call. The problem is that the time consumed by the **Time Manager** and by any interrupt latency (which is not predictable) causes the task to be called at a slightly slower and unpredictable frequency, which drifts over time. In the following figure, a fixed frequency of 1000 microseconds is desired but cannot be achieved because the **Time Manager** overhead and interrupt latency cause a small and unpredictable delay each time the task is reactivated.

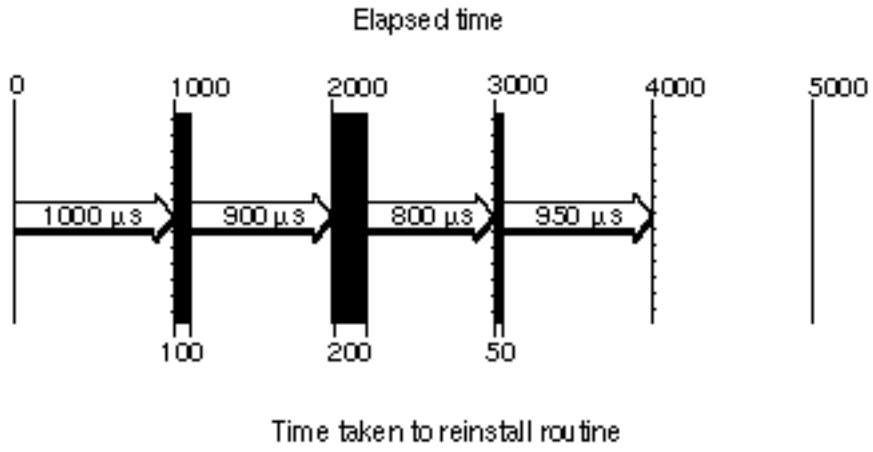


Original and revised Time Managers (drifting, unpredictable frequency)

The extended **Time Manager** solves this problem by allowing you to reinstall a task with an execution time that is relative to the time when the task last expired-not relative to the time when the task is reinstalled. The extended **Time Manager** compensates for the delay between the time when the task last expired and the time at which it was reinstalled, thereby providing a truly drift-free, fixed-frequency timing service.

For example, if an application needs to execute a routine periodically at 1-millisecond intervals, it can reactivate the existing **Time Manager** queue element by calling **PrimeTime** in the task with a specified delay of 1 millisecond. When the **Time Manager** receives this new execution request, it determines how long ago the previous **PrimeTime** task expired and then decrements the specified delay by that amount. For instance, if the previous task expired 100 microseconds ago, then the **Time Manager** installs the new task with a delay of 900 microseconds. This is illustrated in the following figure.

THINK Reference © 1991-1992 Symantec Corporation



The extended Time Manager (drift-free, fixed frequency)

The extended **Time Manager** implements these features by recognizing an expanded task record and providing a new procedure, **InsXTime**. The **Time Manager** task record for the extended **Time Manager** looks like this:

```
struct TMTask {                 // extended Time Manager task record}
QElemPtr    qLink               // next queue entry
Short       qType                // queue type
ProcPtr     tmAddr              // pointer to task
Long        tmCount             // unused time
Long        tmWakeUp            // wakeup time
Long        tmReserved          // reserved for future use
};
```

Once again, your application provides the tmAddr field. You should set tmWakeUp and tmReserved to 0 when you first install an extended **Time Manager** task. The remaining fields are used internally by the **Time Manager**. As in the revised **Time Manager**, the tmCount field holds the time remaining until the scheduled execution of the task (this field is set by **RmvTime**).

The tmWakeUp field contains the time at which the **Time Manager** task specified by tmAddr last executed or contains 0 if it has not yet executed. Its principal intended use is to provide drift-free, fixed-frequency timing services, which are available only when you use the extended **Time Manager** and only when you install **Time Manager** tasks using the new **InsXTime** procedure. When your application installs an extended **Time Manager** task (using the **InsXTime** procedure), the behavior of the **PrimeTime** procedure changes slightly, as described earlier in this section. If the tmWakeUp field is zero when **PrimeTime** is called, the delay parameter to **PrimeTime** is interpreted as relative to the current time (just as in the original **Time Manager**), but the **Time Manager** sets the tmWakeUp field to a nonzero value that indicates when the delay time should expire. When your application calls **PrimeTime** on a **Time Manager** task that has a nonzero value in the tmWakeUp field, the **Time Manager** interprets the specified delay as relative to the time that the last call to **PrimeTime** on this task was supposed to expire.

Note: Nonzero values in tmWakeUp are in a format that is used internally

THINK Reference © 1991-1992 Symantec Corporation

by the **Time Manager** and is subject to change. Your application should never use the value stored in this field and should either set it to 0 or leave it unchanged. When you first create an extended task record, you must ensure that the tmWakeUp field is 0; otherwise, the **Time Manager** may interpret it as a prior execution time.

The extended **Time Manager** allows for a situation that was previously impossible and that may lead to undesirable results. It is possible to call **PrimeTime** with an execution time that is in the past instead of in the future. (With the original and revised Time Managers, only future execution times are possible.) This situation arises when the time in the tmWakeUp field is sometime in the past (which is most common in the tmAddr service routine) and you issue a new **PrimeTime** request with a delay value that is not large enough to cause the execution time to be in the future. This may occur when fixed, high-frequency execution is required and the time needed to process each execution, including the **Time Manager** overhead, is greater than the delay time between requests.

When your application issues a **PrimeTime** request with a tmWakeUp value that would result in a negative delay, the actual delay time is set to 0. The **Time Manager** updates the tmWakeUp field to indicate the time when the task should have awakened (in the past). Because the actual delay time is set to 0, the task executes immediately. If your application continually issues **PrimeTime** requests for times in the past, the **Time Manager** and the tmAddr tasks consume all of the processor cycles. As a result, no time is left for the application to run. This situation is a function of processor speed, so you should test applications that use extended **Time Manager** features on the slowest processors to ensure compatibility. Another solution to this problem is to vary the wakeup frequency according to the processing power of the machine.

Other Time-Related Facilities

The Operating System and Toolbox include several other time-related facilities that complement the services provided by the **Time Manager**. There are three principal facilities: the **TickCount** function, the **Delay** function, and the **Vertical Retrace Manager**. One or more of these services may be more appropriate for your particular timing needs than the **Time Manager**.

THINK Reference © 1991-1992 Symantec Corporation