InsTime Install timer task

#include <Timer.h> **Time Manager**

void **InsTime(***tmTaskPtr* **)**;
QElemPtr *tmTaskPtr* ; address of a 12-byte TMTask structure

Trap macro _InsTime
On entry A0: address of TMTask record
On exit D0: result code

InsTime adds the **Time Manager** task record specified by tmTaskPtr to the **Time Manager** queue. Your application should fill in the tmAddr field of the task record and should set the remaining fields to 0. The tmTaskPtr parameter must point to an original **Time Manager** task record.

With the revised and extended Time Managers, you can set tmAddr to NIL **if** you do not want a task to execute when the delay passed to **PrimeTime** expires. Also, calling **InsTime** with the revised **Time Manager** causes the high-order bit of the qType field to be reset to 0.

InsTime enqueues a data structure in preparation for a subsequent call to **PrimeTime**, which starts the clock ticking on a timer-alarm triggered task. After the alarm goes off, your task is still in the **Time Manager** queue, but it is no longer primed.

Use the **InsXTime** procedure if you want to take advantage of the drift-free, fixed-frequency timing services of the extended **Time Manager**.

tmTaskPtr is the address of a 12-byte TMTask structure. You must set the tmAddr field of this structure to point to the code you wish executed when the alarm is triggered. Other fields are set automatically.

Returns: none

Notes: **Time Manager** alarms have resolution of 1ms (1/1000th of a second). You may enqueue any number of these "wakeup" requests. None takes effect until you call **PrimeTime** and the specified interval elapses.

The code of the timer task is executed at interrupt time, so it must be written with the following constraints:

- It must not call **Memory Manager** functions directly *or indirectly*. You cannot call any Toolbox routine that may move or purge memory.Note: it is OK to call **PrimeTime** to set up for another alarm.
- It must not depend on handles of unlocked memory blocks to be valid.
- It must preserve the values of all registers except A0-A3 and D0-D3.
- If it accesses application globals, it must be sure that the A5 register is valid.

The latter requirement is important. At interrupt time, you cannot depend

on A5 to be the same as when the application is running. Since your application uses A5 to reference its globals, you must set it before accessing application variables. You should use **SetCurrentA5** and **SetA5** to accomplish this. You can't use CurrentA5, since it could contain another application's A5 under MultiFinder. If you are intalling a **Time Manager** task from a code resource or driver, you must save and restore register A4, not A5.

Until System 6.0.3, programmers had to store the application's A5 directly inside the Task's code. Starting with version 6.0.3, the **Time Manager** was rewritten to pass the address of the TMTask in register A1, so you can "piggyback" your global pointer at the end of your TMTask structure. See the example below for details.

Another way to obtain a time slice is to use the Vertical Retrace interrupt (see **VInstall**), or you can write a device driver in which the dNeedTime flag in the driver header is set. In that case, you will get a shot at execution each time any application calls **SystemTask** or **WaitNextEvent**.

Example

```
// An example of the installation of a Time Manager task that changes the value
// of a global variable. Each time that the task is called, the global variable
//aVariable is incremented.  The value of aVariable is printed each time
// through the event loop.
// Clicking the mouse exits the program.

#include <stdio.h>
#include <Timer.h>
// Assumes inclusion of <MacHeaders>

void InstallTMTask (void);
pascal void MyTask (void);
void ToolBoxInit (void);

#define delay  2000

short aVariable = 1;        // the global variable that we will change
                            // in the task

typedef struct {                    // Time Manager information record
    TMTask    atmTask;      // original and revised TMtask record
    long      tmRefCon; // space to pass address of A5 world
} TMInfo;

typedef TMInfo *TMInfoPtr;

pascal TMInfoPtr GetTMInfo (void)
    = 0x2E89;                   // MOVE.L A1,(SP)

TMInfo      myTMInfo;      // a TM information record

pascal void MyTask()            // for revised and extended TMs
{
```

```
    long        oldA5;          // A5 when task is called
    TMInfoPtr recPtr;

    recPtr = GetTMInfo();       // first get your record
    oldA5 = SetA5(recPtr->tmRefCon); //set A5 to app's A5 world

    // do something with the application's globals in here
    aVariable++;

    oldA5 = SetA5(oldA5);           // restore original A5
                                    // and ignore result
    PrimeTime( (QElemPtr) recPtr, delay );
}

void InstallTMTask (void)
{

    myTMInfo.atmTask.tmAddr = MyTask;    // get address of task
    myTMInfo.atmTask.tmWakeUp = 0;       // initialize tmWakeUp
    myTMInfo.atmTask.tmReserved = 0; // initialize tmReserved
    myTMInfo.tmRefCon = SetCurrentA5();// store address of your A5
                                                    //world
    InsTime((QElemPtr) &myTMInfo);              // install the info record
    PrimeTime((QElemPtr) &myTMInfo, delay);// activate the info record
}

void ToolBoxInit ()
{
    InitGraf (&thePort);
    InitFonts ();
    InitWindows ();
    InitMenus ();
    TEInit ();
    InitDialogs (nil);
    InitCursor ();
}


main ()
{
    Boolean done = FALSE;
    EventRecord theEvent;

    ToolBoxInit();
    InstallTMTask ();
    while (!done) {
        WaitNextEvent ( everyEvent, &theEvent, 500, nil );
        printf ("%d\n", aVariable);
        switch (theEvent.what) {
        case mouseDown:
            done = TRUE;
            RmvTime((QElemPtr) &myTMInfo);
            break;
        }
    }
```

```
}
```