

TECHNOTE:

Inside Macintosh: Devices, Power Manager Addenda

By Vinnie Moscaritolo

<<http://webstuff.apple.com/~vinnie/>>@apple.com
vinnie@apple.com

Apple Developer Technical Support (DTS)

This Technote discusses addenda to the *Inside Macintosh: Devices, Power Manager* Chapter.

This Note is intended for Macintosh developers who are using the Power Manager Dispatch routines.

Topics

- PMFeatures, April 1996
- Startup Time Structure, April 1996
- GetStartupTimer, April 1996
- SetStartupTimer, April 1996
- State Change Notification Queue Structure, April 1996
- PMgrStateQInstall, April 1996
- PMgrStateQRemove, April 1996

- State Change Queue Notification Function, April 1996
- UpdateSystemActivity, April 1996
- GetLastActivity, April 1996
- DelaySystemIdle, April 1996

Addenda to the Power Manager Dispatch Routines

PMFeatures

One additional bit field: `hasStartupTimer` has been added to the possible `PMFeature` return values:

Bit Name	Bit Number	Description
<code>hasStartupTimer</code>	10	The startup timer is supported.

Startup Time Structure

The startup time structure used by the `GetStartupTimer` and `SetStartupTimer` functions is defined by the `StartupTime` data type.

```
typedef struct StartupTime {
    unsigned longstartTime;    /* startup time as number of seconds since
                               midnight, January 1, 1904 */
    Boolean      startEnabled; /* 1=enable startup timer, 0=disable timer*/
    SInt8       filler;
} StartupTime;
```

GetStartupTimer

You can use the `GetStartupTimer` function to find out when the computer will start up from power off mode.

```
void GetStartupTimer(StartupTime *theTime);
```

theTime A pointer to a `StartupTime` structure, which specifies whether the timer is enabled or disabled and the time at which the startup timer is set to start up the computer.

DESCRIPTION

The `GetStartupTimer` function returns the time when the computer will start up from a power off mode.

If the computer doesn't support the startup timer, `GetStartupTimer` returns a value of 0.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerManagerDispatch` (\$A09E). The selector value is for `GetStartupTimer` is 38 (\$26) in the low word of register D0. The pointer to `StartupTime` is passed in register A0.

SEE ALSO

The `StartupTime` structure is described in “Startup Time Structure”.

SetStartupTimer

You can use the `SetStartupTimer` function to set the time when the computer will start up from power off mode.

```
void SetStartupTimer(StartupTime *theTime);
```

theTime A pointer to a `StartupTime` structure, which specifies whether to enable or disable the timer, and the time at which the startup timer is set to start up the computer.

DESCRIPTION

The `SetStartupTimer` function sets the time when the computer will start up from a power off mode and enables or disables the timer. On a computer that doesn't support the startup timer, `SetStartupTimer` does nothing.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerManagerDispatch` (\$A09E). The selector value is for `SetStartupTimer` is 39 (\$27) in the low word of register D0. The pointer to `StartupTime` is passed in register A0.

SEE ALSO

The `StartupTime` structure is described in “Startup Time Structure”.

State Change Notification Queue Structure

The state change notification queue structure used by the `PMgrStateQInstall` and `PMgrStateQInstall` functions is defined by the `PMgrQueueElement` data type.

```
#define PMgrStateQType 'PM'

struct PMgrQueueElement {
    Ptr          *pmQLink;      /* pointer to next queue element*/
    short        pmQType;      /* queue type (must be
                                PMgrStateQType)*/
    short        pmFlags;      /* as defined below*/
    long         pmNotifyBits;  /* bitmap of desired notifications*/
    StateNotifyProc pmProc;    /* pointer to routine to call*/
    long         pmUser;       /* user-defined private storage*/
} PMgrQueueElement;
```

The values of the bits in the `pmFlags` field are as follows:

Bit Name	Bit Number	Description
<code>sleepTimeout</code>	0	The sleep timer value has changed.
<code>sleepEnable</code>	1	Sleep control has been enabled/disabled
<code>hardDiskTimeout</code>	2	The hard disk timer has changed.
<code>hardDiskSpindown</code>	3	The hard disk spindown state has changed.
<code>dimmingTimeout</code>	4	The screen dimming timer has changed.
<code>dimmingEnable</code>	5	Screen dimming has been enabled or disabled
<code>diskModeAddress</code>	6	The SCSI disk mode address has changed
<code>processorCycling</code>	7	Processor cycling has been enabled or disabled

Bit Name	Bit Number	Description
processorSpeed	8	The processor speed has changed
wakeupTimer	9	The wakeup timer has changed.
startupTimer	10	The startup timer has changed.
hdPowerRemoved	11	The hard disk power has been removed by the user.

PMgrStateQInstall

You can use the `PMgrStateQInstall` function to notify your software when Power Manager state configuration parameters have changed .

```
OSErr PMgrStateQInstall(PMgrStateQElement *theElement);
```

`theElement` A pointer to an element for the state change notification queue.

DESCRIPTION

The `PMgrStateQInstall` function installs an element into the state configuration change queue to provide notification to your software when certain Power Manager state configuration parameters have changed.

When a requested configuration parameter has changed, the software calls the routine pointed to by the `pmProc` field so that it can do any special processing. The routine is passed a pointer to its queue element so that, for example, the routine can reference its variables.

Before calling `PMgrStateQInstall`, the calling program must set the `pmQType` field to `PmgrStateQtype` or the queue element won't be added to the queue and `PMgrStateQInstall` will return an error.

SPECIAL CONSIDERATIONS

The `pmNotifyBits` field may be modified by the `PMgrStateQInstall` if it requests notification for a feature that this computer does not support. In this case your software may remove the element.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerManagerDispatch` (\$A09E). The selector value for `PMgrStateQInstall` is 34 (\$22) in the low word of register D0. The pointer to `PMgrStateQElement` is passed in register A0. The result code is returned in the low word of register D0.

RESULT CODE

<code>noErr</code>	0	No error
--------------------	---	----------

SEE ALSO

The `PMgrStateQElement` structure is described in “State Change Notification Queue Structure.”

The application-defined state change notification function is described in “State Change Notification Function.”

PMgrStateQRemove

You can use the `PMgrStateQRemove` function to discontinue notification of your software when Power Manager state configuration parameters have changed.

```
OSErr PMgrStateQRemove(PMgrStateQElement *theElement);
```

<code>theElement</code>	A pointer to the element for the state change notification queue that you wish to remove.
-------------------------	---

DESCRIPTION

The `PMgrStateQRemove` function removes a queue element installed by `PMgrStateQInstall`. If the `pmQType` field of the queue element is not set to `PMgrStateQtype`, `PMgrStateQInstall` will return an error.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerManagerDispatch` (\$A09E). The selector value is for `PMgrStateQRemove` is 35 (\$23) in the low word of register D0. The pointer to `PMgrStateQElement` is passed in register A0. The result code is returned in the low word of register D0.

RESULT CODE

noErr 0 No error

SEE ALSO

The `PMgrStateQElement` structure is described in “State Change Notification Queue Structure”.

The application-defined state change notification function is described in “State Change Notification Function.”

State Change Notification Function

A state change notification function can perform any operation that you require when certain Power Manager state configuration parameters have changed .

```
pascal void MyStateNotificationProc (PMgrStateQElement *theElement);
```

`theElement` A pointer to the element in the configuration change queue that was used to install this function.

DESCRIPTION

Your state change notification function is called by the Power Manager when certain configuration parameters have changed. It is called, for instance, when the sleep timer value or processor speed have changed. You can determine which of the configuration parameters have changed by examining the bit map in the `pmNotifyBits` field of the `PMgrStateQElement` that was passed to your function.

SEE ALSO

The `PMgrStateQElement` structure is described in “State Change Notification Queue Structure”.

The `PMgrStateQInstall` function and the `PMgrStateQRemove` function.

UpdateSystemActivity

You can use the `UpdateSystemActivity` function to notify the Power Manager that activity has taken place .

```
void UpdateSystemActivity(short activityType);
```

`activityType` A value indicating the type of activity that has occurred See the description below for the meaning of this field.

DESCRIPTION

The `UpdateSystemActivity` function is used to notify the Power Manager that activity has taken place and the timers used to measure idle time should be updated to the time of this call. This function can be used by device drivers to prevent the computer from entering a low power mode while critical activity is taking place on a particular device. The function is passed a parameter indicating the type of activity that has occurred.

This function is slightly different from `DelaySystemIdle`, which should be used to prevent sleep or idle during a critical section. `UpdateSystemActivity` simply updates the tick count for the activity type selected. Conversely, `DelaySystemIdle` actually moves the counter to some number of ticks into the future, which allows the caller to go off and do something without fear of idling.

The valid types of activity are:

Value Name	Value	Description
OverallAct	0	general type of activity
UsrActivity	1	user activity (i.e.keyboard or mouse)
NetActivity	2	interaction with network(s)
HDAActivity	3	hard disk or storage device in use

SPECIAL CONSIDERATIONS

In general, device drivers should make this call to notify the Power Manager of system activity on their particular device. For example, input device drivers would call `UpdateSystemActivity(UsrActivity)` while a storage device driver would call `UpdateSystemActivity(HDAActivity)`.

If a device driver registers with the power manager state change queue and implements its own power management timing facilities, it does not need to call `UpdateSystemActivity`. For example, a media bay hard disk on a PowerBook should register for `sleeptimeout`, `sleepenabled`, `spindownenabled`, and `hdttimeout` change notifications. It then can maintain its own timer. This allows the internal hard disk and the media bay hard disk to spin down independently. By not calling `UpdateSystemActivity`, it won't unnecessarily keep the internal drive powered. It should also be installed in the sleep queue so it can refuse sleep requests if activity is occurring on its associated device.

IMPORTANT

Calling `UpdateSystemActivity` more than necessary will cause delays in the invocation of power saving features which will reduce battery life and overall power efficiency.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerManagerDispatch` (\$A09E). The selector value for `UpdateSystemActivity` is 36 (\$24) in the low word of register D0. The `activityType` is passed in the high word of D0.

SEE ALSO

The `GetLastActivity` function.

GetLastActivity

You can use the `GetLastActivity` function to find out when the last time a particular activity took place.

```
OSErr GetLastActivity(ActivityInfo *theActivity());
```

`theActivity` A pointer to an `ActivityInfo` structure, which specifies the type of activity and when that activity last occurred.

DESCRIPTION

The `GetLastActivity` function returns the time in ticks of the last occurrence of the specified activity type. This function can be used by software that needs to track power management activity. The activity type and time are specified in the following structure:

```
struct ActivityInfo {
    short          activityType;    /* same selectors as
                                   UpdateSystemActivity */
    unsigned long  activityTime;    /*time of last event of selected
                                   type as number of seconds since
                                   midnight, January 1, 1904 */
} ActivityInfo;
```

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerManagerDispatch` (\$A09E). The selector value for `GetLastActivity` is 40 (\$28) in the low word of register D0. The pointer to the `ActivityInfo` structure is passed in register A0.

SEE ALSO

The `UpdateSystemActivity` function.

DelaySystemIdle

You can use the `DelaySystemIdle` function to control power cycling.

```
void DelaySystemIdle();
```

DESCRIPTION

The `DelaySystemIdle` function delays power cycling and/or sleep to a time in the near future. This call should only be used when timing-sensitive I/O operations (e.g. Read/Write) need to occur without interruption from the Power Manager and the power cycling wait time or sleep timeout may not be long enough to ensure that this occurs.

SPECIAL CONSIDERATIONS

Care should be taken to make this call [only when absolutely necessary](#) since it can affect power efficiency.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerManagerDispatch` (\$A09E). The selector value is for `DelaySystemIdle` is 37 (\$25) in the low word of register D0.

[Acknowledgments](#)

Thanks to Cameron Birse, Paul Black, Russ Emmons, Mike Hanlon, and Mark Pontarelli.

