This chapter describes how you can use the Notification Manager to inform users of significant occurrences in applications that are running in the background or in software that is largely invisible to the user. This software includes device drivers, vertical blanking (VBL) tasks, Time Manager tasks, completion routines, and desk accessories that operate behind the scenes. It also includes code that executes during the system startup sequence, such as code contained in `'INIT'` resources.

The Notification Manager is available in system software versions 6.0 and later. You can use the `Gestalt` function to determine whether the Notification Manager is present. See the chapter "Gestalt Manager" in *Inside Macintosh: Operating System Utilities* for complete details on using `Gestalt`.

You need to read this chapter if your application, desk accessory, or device driver might need to notify the user of some occurrence while it is running in the background or is otherwise invisible to the user. You also need to read this chapter if you want to write `'INIT'` resources that might need to inform the user of important occurrences during their execution at system startup time.

About the Notification Manager

The Notification Manager provides a notification service. It allows software running in the background (or otherwise unseen by the user) to communicate information to the user. For example, applications that manage lengthy background tasks (such as printing many documents or transferring large amounts of data to other machines) might need to inform the user that the operation is complete. These applications cannot use the standard methods of communicating with the user, such as alert or dialog boxes, because such windows might easily be obscured by the windows of other applications. Moreover, even if those windows are visible, the background application cannot be certain that the user is aware of the change. A more reliable method is needed to manage the communication between a background application and the user, who might be awaiting the completion of the background task while running some other application in the foreground.

In the same way, relatively invisible operations such as Time Manager tasks, VBL tasks, or device drivers might need to inform the user that some previously started routine is complete or perhaps that some error has rendered further execution undesirable or impossible.

In all these cases, the communication generally needs to occur in one direction only, from the background application (or task, or driver) to the user. The Notification Manager, included in system software versions 6.0 and later, allows you to alert the user by posting a **notification,** which is an audible or visible indication that your application (or other piece of software) requires the user's attention. You post a notification by issuing a **notification request** to the Notification Manager, which places your request in a queue. When your request reaches the top of the queue, the Notification Manager posts a notification to the user.

You can request three types of notification:

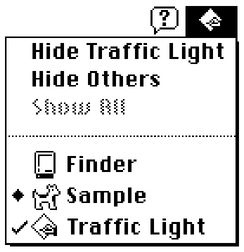
- **Polite notification.** A small icon blinks, by periodically alternating with the Apple menu icon (the Apple logo) or the Application menu icon in the menu bar.
- **Audible notification.** The Sound Manager plays the system alert sound or a sound contained in an `'snd '` resource.
- **Alert notification.** An alert box containing a short message appears on the screen. The user must dismiss the alert box (by clicking the OK button) before foreground processing can continue.

These types of notification are not mutually exclusive; for example, an application can request both audible and alert notifications. Moreover, if the requesting software is listed in the Application menu (and hence represents a process that is loaded into memory), you can instruct the Notification Manager to place a diamond-shaped mark next to the name of the requesting process. The mark is usually intended to prompt the user to switch the marked application into the foreground. Finally, you can request that the Notification Manager execute a **notification response procedure,** which is executed as the final step in a notification.

In short, a notification consists of one or more of five possible actions. If you request more than one action, they occur in the following order:

1. A diamond-shaped mark appears next to the name of your application in the Application menu, as illustrated in Figure 5-1. Note that the diamond is present only when your application is in the background (because the diamond is replaced by a checkmark if your application is the active application). In Figure 5-1, the Traffic Light application is the active application.

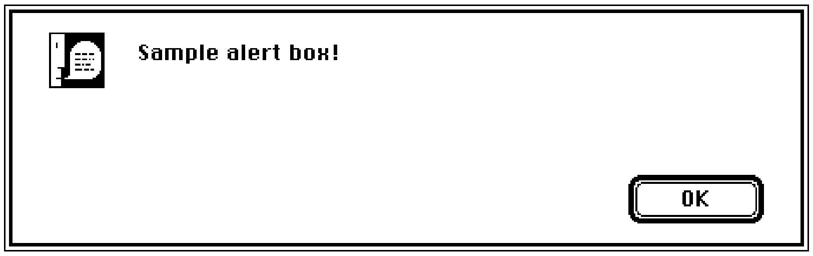
Figure 5-1 A notification in the Application menu



2. A small icon blinks, alternating with either the Apple menu icon or the Application menu icon in the menu bar. Typically, the small icon is your application's small icon. Because several applications can post notifications, there might be a series of small icons blinking in the menu bar. The location of each blinking icon varies according to the posting application's mark (if any). If your application is marked with a diamond (or a checkmark) in the Application menu, the icon blinks above the Application menu; otherwise, the icon blinks above the Apple menu.

3. The Sound Manager plays a sound. Your application can supply its own sound (by passing the Notification Manager a handle to an `'snd '` resource loaded into memory) or request that the Sound Manager use the user's system alert sound.
4. An alert box like the one in Figure 5-2 appears, and the user dismisses it. Your application specifies the text in the alert box.

Figure 5-2 A notification alert box


Sample alert box!

5. A response procedure is executed. You can use the response procedure to remove the notification request from the queue or perform other processing.

The mark in the Application menu and the blinking small icon remain until the requesting application removes the notification request from the queue. However, the sound and the alert box are presented only once, if at all.

Any applications, desk accessories, tasks, routines, or drivers can use the Notification Manager, whether they are running in the background or not. It is especially useful for background applications, such as the PrintMonitor application. (The system alarm, which is called by the Alarm Clock desk accessory, also uses the Notification Manager.) Foreground applications can, however, use the Notification Manager to achieve effects (such as the blinking small icon) that are otherwise more difficult to create. For the same reasons, the Notification Manager can be useful even to applications that might be executing in a Finder-only environment under system software version 6.0.

The Notification Manager provides applications with a standard user interface for notifying the user of significant events. The following three-level notification strategy for communicating with the user is recommended:

1. Display a diamond next to the name of the application in the Application menu.
2. Insert a small icon into the list of icons displayed alternately with the Apple menu icon or the Application menu icon in the menu bar, and display a diamond next to the name of your application in the Application menu.
3. Display a diamond, insert a small icon, and display an alert box to notify the user that something needs to be done.

Ideally, the user should be allowed to set the desired level of notification. The suggested default level of notification is level 2. In levels 2 and 3, you might also play a sound, but the user should have the ability to turn the sound off. In addition, a user should have the ability to turn off background notification altogether, except when damage might occur or data might be lost.

Note
This suggested notification strategy may not be appropriate for your application. Notifications posted by system software might not follow these guidelines. ◆

Each application, desk accessory, and device driver can issue any number of notification requests. Each requested notification is presented separately to the user. For this reason, avoid posting multiple notification requests for the same occurrence. Depending on the method of notification you specify, multiple requests might result in an annoying number of notification sounds or many alert boxes that the user must dismiss before continuing.

Note that the Notification Manager provides a one-way communications path from an application to the user. There is no provision for carrying information back from the user to the requesting application, although it is possible for the requesting application to determine if the notification was received. If you require this secondary communications link, do not use the Notification Manager. Instead, you should wait until the user switches your application into the foreground and then use standard means (for example, a dialog box) to obtain the required information.

Using the Notification Manager

To issue a notification to the user, you need to create a notification request and install it in the notification queue. The Notification Manager interprets the request and presents the notification to the user at the earliest possible time. After you have notified the user in the desired manner (that is, placed a diamond mark in the Application menu, added a small blinking icon to the menu bar, played a sound, or displayed an alert box), you might want the Notification Manager to call a response procedure. The response procedure is useful for determining that the user has indeed seen the notification or for reacting to the successful posting of the notification. Eventually, you need to remove the notification request from the notification queue; you can do this in the response procedure or when your application returns to the foreground.

The Notification Manager is automatically initialized at system startup time. It includes two functions, one that allows you to install a request into the notification queue and one that allows you to remove a request from that queue.

Creating a Notification Request

Information describing each notification request is contained in the **notification queue,** which is a standard operating-system queue (as described in the chapter “Queue Utilities” in *Inside Macintosh: Operating System Utilities*). Each entry in the notification queue is a **notification record**—a static and nonrelocatable record of type `NMRec`. When installing a request in the notification queue, your application must supply a pointer to a notification record that indicates the type of notification you desire. Here is the NMRec data structure:

```
TYPE NMRec =
   RECORD
      qLink:      QElemPtr;   {next queue entry}
      qType:      Integer;    {queue type}
      nmFlags:    Integer;    {reserved}
      nmPrivate:  LongInt;    {reserved}
      nmReserved: Integer;    {reserved}
      nmMark:     Integer;    {item to mark in menu}
      nmIcon:     Handle;     {handle to icon}
      nmSound:    Handle;     {handle to sound resource}
      nmStr:      StringPtr;  {string to appear in alert box}
      nmResp:     ProcPtr;    {pointer to response procedure}
      nmRefCon:   LongInt;    {for application's use}
   END;
```

To set up a notification request, you need to fill in the fields qType, nmMark, nmIcon, nmSound, nmStr, nmResp, and nmRefCon. The remaining fields of this record are used internally by the Notification Manager or are reserved for use by Apple Computer, Inc.

Field descriptions

qLink	Points to the next element in the queue. This field is used internally by the Notification Manager.
qType	Indicates the type of queue. You should set this field to the value ORD(nmType), which is 8.
nmFlags	Reserved for use by Apple Computer, Inc.
nmPrivate	Reserved for use by Apple Computer, Inc.
nmReserved	Reserved for use by Apple Computer, Inc.
nmMark	Indicates whether to place a diamond-shaped mark next to the name of the application in the Application menu. If the value of nmMark is 0, no such mark appears. If the value of nmMark is 1, the mark appears next to the name of the calling application. If the value of nmMark is neither 0 nor 1, it is interpreted as the reference number of a desk accessory. An application should pass 1, a desk accessory should pass its own reference number, and a driver or a detached background task (such as a VBL task or Time Manager task) should pass 0.
nmIcon	Contains a handle to a small icon that is to blink periodically in the menu bar. If the value of nmIcon is NIL, no icon appears in the menu bar. This handle must be valid at the time that the notification occurs; it does not need to be locked, but it must be nonpurgeable.
nmSound	Contains a handle to a sound resource to be played with SndPlay. If the value of nmSound is NIL, no sound is produced. If the value

	of `nmSound` is –1, then the system alert sound plays. This handle does not need to be locked, but it must be nonpurgeable.
`nmStr`	Points to a string that appears in the alert box. If the value of `nmStr` is `NIL`, no alert box appears. Because the Notification Manager does not make a copy of this string, your application should not release this memory until it removes the notification request.
`nmResp`	Points to a response procedure. If the value of `nmResp` is `NIL`, no response procedure is executed when the notification is posted. If the value of `nmResp` is –1, then a predefined procedure removes the notification request immediately after it has completed.
`nmRefCon`	A long integer available for your application's own use.

Listing 5-1 illustrates how to set up a notification record. In this listing, `gMyNotification` is a global variable of type `NMRec` and `gText` is a global variable of type `Str255`.

Listing 5-1 Setting up a notification record

```
VAR
   myResNum:         Integer;    {resource ID of small icon}
   myResHand:        Handle;     {handle to small icon resource}
BEGIN
   myResNum := 1234;             {resource ID in resource fork}
   myResHand := GetResource('SICN', myResNum);
                                 {get icon from resource fork}
   gText := 'Sample Alert Box';  {set message for alert box}
   WITH gMyNotification DO
   BEGIN
      qType := ORD(nmType);      {set queue type}
      nmMark := 1;               {put mark in Application menu}
      nmIcon := myResHand;       {blinking icon}
      nmSound := Handle(-1);     {play system alert sound}
      nmStr := @gText;           {display alert box}
      nmResp := NIL;             {no response procedure}
      nmRefCon := 0;             {not needed here}
   END;
END;
```

This notification record requests all three types of notification—polite (blinking small icon), audible (system alert sound), and alert (alert box). In addition, the diamond appears in front of the application's name in the Application menu. In this case, the small icon has resource ID 1234 of type `'SICN'` in the application's resource fork.

Defining a Response Procedure

The `nmResp` field of the notification record contains the address of a response procedure executed as the final stage of a notification. If no processing is necessary in response to the notification, then you can supply the value `NIL` in that field. If you supply the address of your own response procedure in the `nmResp` field, the Notification Manager passes it one parameter, a pointer to your notification record. For example, this is how you would declare a response procedure having the name `MyResponse`:

```
PROCEDURE MyResponse (nmReqPtr: NMRecPtr);
```

When the Notification Manager calls this response procedure, it does not set up the A5 register or application-specific system global variables for you. If you need to access your application's global variables, you should save its A5 value in the `nmRefCon` field. See the chapter "Memory Management Utilities" in the book *Inside Macintosh: Memory* for more information on saving and restoring the A5 world.

Response procedures should never cause anything to be drawn on the screen or otherwise affect the human interface. Rather, you should use them simply to remove notification requests from the notification queue and free any memory. If you specify the special `nmResp` value of –1, the Notification Manager removes the queue element from the queue automatically, and you don't have to do it yourself. You have to pass your own response routine, however, if you need to do anything else in the response procedure, such as free the memory block containing the queue element or set an application global variable indicating that the notification was received.

If you use audible or alert notifications, you should probably set `nmResp` to –1 to remove the notification record from the queue as soon as the sound ends or the user dismisses the alert box. However, if either `nmMark` or `nmIcon` has a nonzero value, you should not set `nmResp` to –1 (because the Notification Manager would remove the diamond mark or the small icon before the user could see it). Note that when the value of `nmResp` is –1, the Notification Manager does not free the memory block containing the queue element; it merely removes that element from the notification queue.

Because the execution of the response procedure is the last step in the notification process, your application can determine whether the notification was posted by examining a global variable that you set in the response procedure. In addition, to determine that the user has actually received the notification, you need to request an alert notification. This is necessary because the response procedure is executed only after the user clicks the OK button in the alert box.

Installing a Notification Request

To add a notification request to the notification queue, call the `NMInstall` function. For example, you can install the notification request defined in Listing 5-1 with the following line of code:

```
myErr := NMInstall(@gMyNotification);     {install request}
```

If the call to `NMInstall` returns an error, then you cannot install the notification request in the notification queue. In that case, your application should wait for the user to switch it to the foreground before doing further processing. While waiting for a resume event, your application should take care of other events, such as updates. Note, however, that `NMInstall` fails only if it is passed invalid information, namely, the wrong value for `qType`.

You can install notification requests at any time, even when the system is executing `'INIT'` resources as part of the system startup sequence. If you need to notify the user of some important occurrence during the execution of your `'INIT'` resource, use the Notification Manager to install a request in the notification queue. The system notifies the user after the startup process completes, that is, when the normal event mechanism begins. This saves you from having to interrupt the system startup sequence with dialog or alert boxes and results in a cleaner and more uniform startup appearance.

Removing a Notification Request

To remove a notification request from the notification queue, call the `NMRemove` function. For example, you can remove a notification request with this code:

```
myErr := NMRemove(@gMyNotification);      {remove request}
```

You can remove requests at any time, either before or after the notification actually occurs. Note that requests already issued by the Notification Manager are not automatically removed from the queue.

Notification Manager Reference

This section describes the routines that are specific to the Notification Manager. It also describes the application-defined notification response procedure.

Notification Manager Routines

The Notification Manager includes two functions, one to install a notification request and one to remove it.

NMInstall

To install a notification request, use the `NMInstall` function.

```
FUNCTION NMInstall (nmReqPtr: NMRecPtr): OSErr;
```

`nmReqPtr`	A pointer to a notification record.

DESCRIPTION

The `NMInstall` function adds the notification request specified by the `nmReqPtr` parameter to the notification queue and returns a result code.

SPECIAL CONSIDERATIONS

Because `NMInstall` does not move or purge memory, you can call it from completion routines or interrupt handlers as well as from the main body of an application and from the response procedure of a notification request.

ASSEMBLY-LANGUAGE INFORMATION

The registers on entry and exit for `NMInstall` are

Registers on entry

A0	Address of `NMRec` record

Registers on exit

D0	Result code

RESULT CODES

`noErr`	0	No error
`nmTypErr`	–299	Invalid `qType` value (must be `ORD(nmType)`)

NMRemove

To remove a notification request, use the `NMRemove` function.

```
FUNCTION NMRemove (nmReqPtr: NMRecPtr): OSErr;
```

`nmReqPtr`	A pointer to a notification record.

DESCRIPTION

The `NMRemove` function removes the notification request identified by the `nmReqPtr` parameter from the notification queue and returns a result code.

SPECIAL CONSIDERATIONS

Because `NMRemove` does not move or purge memory, you can call it from completion routines or interrupt handlers as well as from the main body of an application and from the response procedure of a notification request.

ASSEMBLY-LANGUAGE INFORMATION

The registers on entry and exit for `NMRemove` are

Registers on entry

A0	Address of `NMRec` record

Registers on exit

D0	Result code

RESULT CODES

`noErr`	0	No error
`qErr`	–1	Not in queue
`nmTypErr`	–299	Invalid `qType` (must be `ORD(nmType)`)

Application-Defined Routine

The Notification Manager allows you to define a notification response procedure.

Notification Response Procedures

You pass the address of an application-defined notification response procedure in the `nmResp` field of a notification record.

MyResponse

If desired, you can specify the address of a completion or response procedure that is executed as the last stage in a notification. The response procedure should have this syntax:

```
PROCEDURE MyResponse (nmReqPtr: NMRecPtr);
```

`nmReqPtr`	A pointer to a notification record.

DESCRIPTION

The `nmResp` field of the notification record contains the address of a response procedure executed as the final stage of a notification. If no processing is necessary in response to the notification, then you can supply the value `NIL` in that field. If you supply the address of your own response procedure in the `nmResp` field, the Notification Manager passes it one parameter, a pointer to your notification record.

SEE ALSO

For more details on a response procedure, see “Defining a Response Procedure” on page 5-9.

Summary of the Notification Manager

Pascal Summary

Constant

```
CONST
   nmType            = 8;              {queue type of notification queue}
```

Data Types

```
TYPE NMRec =
   RECORD
      qLink:         QElemPtr;         {next queue entry}
      qType:         Integer;          {queue type}
      nmFlags:       Integer;          {reserved}
      nmPrivate:     LongInt;          {reserved}
      nmReserved:    Integer;          {reserved}
      nmMark:        Integer;          {item to mark in menu}
      nmIcon:        Handle;           {handle to icon}
      nmSound:       Handle;           {handle to sound resource}
      nmStr:         StringPtr;        {string to appear in alert box}
      nmResp:        ProcPtr;          {pointer to response procedure}
      nmRefCon:      LongInt;          {for application's use}
   END;

   NMRecPtr = ^NMRec;
```

Notification Manager Routines

```
FUNCTION NMInstall          (nmReqPtr: NMRecPtr): OSErr;
FUNCTION NMRemove           (nmReqPtr: NMRecPtr): OSErr;
```

Application-Defined Routine

```
PROCEDURE MyResponse        (nmReqPtr: NMRecPtr);
```

C Summary

Constant

```
enum {nmType         = 8};            /*queue type of notification queue*/
```

Data Types

```
typedef pascal void (*NMProcPtr)(struct NMRec *);

struct NMRec {
      QElemPtr       qLink;             /*next queue entry*/
      short          qType;             /*queue type*/
      short          nmFlags;           /*reserved*/
      long           nmPrivate;         /*reserved*/
      short          nmReserved;        /*reserved*/
      short          nmMark;            /*item to mark in menu*/
      Handle         nmIcon;            /*handle to icon*/
      Handle         nmSound;           /*handle to sound resource*/
      StringPtr      nmStr;             /*string to appear in alert box*/
      NMProcPtr      nmResp;            /*pointer to response procedure*/
      long           nmRefCon;          /*for application's use*/
};

typedef struct NMRec NMRec;
typedef NMRec *NMRecPtr;
```

Notification Manager Routines

```
pascal OSErr NMInstall      (NMRecPtr nmReqPtr);
pascal OSErr NMRemove       (NMRecPtr nmReqPtr);
```

Application-Defined Routine

```
pascal void MyResponse      (NMRecPtr nmReqPtr);
```

Result Codes

`noErr`	0	No error
`qErr`	–1	Not in queue
`nmTypErr`	–299	Invalid `qType` value (must be `ORD(nmType)`)