**Macintosh Technical Notes**

#184: Notification Manager

See also: *Inside Macintosh*, The Operating System Utilities
Technical Note #129: _SysEnvirons: System 6.0 and Beyond
Technical Note #180: MultiFinder Miscellanea

Written by: Darin Adler April 2, 1988
Modified by: Rich Collyer October 1, 1988

This Technical Note describes the Notification Manager, the part of the operating system that lets an application, desk accessory, or driver alert the user.

**Changes since April 2, 1988:** Added sample MPW C 3.0 code.

The Notification Manager, in System Software version 6.0 and later, provides the user with an asynchronous “notification” service. The Notification Manager is especially useful for background applications under MultiFinder; the PrintMonitor application and the system alarm (set by the Alarm Clock desk accessory) both use its services.

Each application, desk accessory, or device driver can queue any number of notifications. For this reason, you should try to avoid posting multiple notifications since each one will be presented separately to the user (i.e., “you have mail,” “you have mail,” etc.).

The Notification Manager queue contains information describing each notification request; you supply a pointer to a queue element describing the type of notification you desire. The Notification Manager queue is a standard Macintosh queue, as described in the Operating System Utilities chapter of *Inside Macintosh*, Volume II. Each entry in the Notification Manager queue has the following structure:

```
TYPE NMRec = RECORD
               qLink:        QElemPtr;      {next queue entry}
               qType:        INTEGER;       {queue type -- ORD(nmType) = 8}
               nmFlags:      INTEGER;       {reserved}
               nmPrivate:    LONGINT;       {reserved}
               nmReserved:   INTEGER;       {reserved}
               nmMark:       INTEGER;       {item to mark in Apple menu}
               nmSIcon:      Handle;        {handle to small icon}
               nmSound:      Handle;        {handle to sound record}
               nmStr:        StringPtr;     {string to appear in alert}
               nmResp:       ProcPtr;       {pointer to response routine}
               nmRefCon:     LONGINT;       {for application use}
             END;
```

To use the Notification Manager, you must also use `_SysEnvirons` (discussed in *Inside Macintosh*, Volume V and Technical Note #129) to test the System Software version. If the System Software is not current and the Notification Manager routines are not present, display an alert to inform the user that your application requires System Software version 6.0 or newer, then exit.

## Using the Notification Manager

Your program can request three types of notifications:

- polite notification: a small icon that periodically appears in rotation with the apple in the menu bar;
- audible notification: a sound to be played by the Sound Manager;
- alert notification: an dialog box containing a short message.

In addition, you can place a diamond mark next to the name of the requesting desk accessory or application in the Apple menu.

After you have notified the user as you feel necessary (placed a diamond mark in the Apple menu, added a small icon to the list of icons which rotate with the apple in the menu bar, played a sound, and presented the user with a dialog box to acknowledge), you should call a response procedure.

### How the Notification Manager Handles Notifications

When the Notification Manager handles a notification, it does one or more of the following (in this order):

- puts a diamond mark next to the requesting application or desk accessory in the Apple menu
- adds a small icon to the list of icons which rotate with the apple in the menu bar
- plays a specified sound
- presents a dialog box for the user to acknowledge and dismiss
- calls the response procedure

At this point, the diamond mark in the Apple menu and the icon rotating with the apple in the menu bar will remain until your application removes the notification request from the queue. The Notification Manager only presents the sound and dialog box once.

### Creating a Notification Request

To create a notification request, you must set up a `NMRec` with all the information about the notification you want:

`nmMark` contains 0 to not place a mark in the Apple menu, 1 to mark the current application, or the `refNum` of a desk accessory to mark that desk accessory. An application should pass 1, a desk accessory should pass is its own `refNum`, and a VBL task should pass 0.

`nmSIcon` contains `NIL` for no icon in the menu bar, or a handle to a small icon to rotate with the apple. (A small icon is a 16 x 16 bit map, often stored in an `'SICN'` resource.) This handle does not need to be locked, but it must be non-purgeable.

`nmSound` contains `NIL` to use no sound, –1 to use the system beep sound, or a handle to a sound record which can be played with `_SndPlay`. This handle does not need to be locked, but it must be non-purgeable.

`nmStr` contains `NIL` for no alert, or a pointer to the string to appear in the alert.

`nmResp` contains `NIL` for no response procedure, –1 for a predefined procedure that removes the request immediately after it is completed, or a pointer to a procedure which takes one parameter, a pointer to your queue element. For example, the following is how you would declare it if it were named MyResponse:

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

Note that when the Notification Manager calls your response procedure, it does not set up `A5` and low-memory globals for you. If you need to access your application's globals, you should save your application's `A5` in the `nmRefCon` field as discussed below.

Response procedures should never draw or do "user interface" things. You should wait until the user brings the application or desk accessory to the front before responding to the user. Some good ways to use the response procedure are to dequeue and deallocate your Notification Manager queue element or to set an application global (being careful about `A5`), so the application knows when the user receives the notification.

You should probably use an `nmResp` of –1 with audible and alert notifications to remove the notification as soon as the sound has finished or the user has dismissed the dialog. You should not use an `nmResp` of –1 with an `nmMark` or an `nmSIcon` because the Notification Manager would remove the diamond mark or small icon before the user could see it. Note that an `nmResp` of –1 does not deallocate the memory block containing the queue element, it only removes it from the notification queue.

You can also use `nmRefCon`; one convenient use us putting your application's `A5` in this field so the response procedure can access application globals. For more information about this technique, refer to the section about VBL tasks in Technical Note #180.

## Notification Manager Routines

The system automatically initializes the Notification Manager when it boots. To add a notification request to the notification queue, call _NMInstall. When your application no longer wants a notification to continue, it can remove the request by calling _NMRemove. Neither _NMInstall nor _NMRemove move or purge memory, and you can call either of them from completion routines or interrupt handlers, as well as from the main body of an application and the response procedure of a notification request.

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

      Trap macro     _NMInstall ($A05E)
      On entry       A0: theNMRec (pointer)
      On exit        D0: result code (word)
```

_NMInstall adds the notification request specified by nmReqPtr to the notification queue and returns one of the following result codes:

```
      Result codes   noErr                No error
                     nmTypErr (-299)      qType field is not ORD(nmType)
```

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

      Trap macro     _NMRemove ($A05F)
      On entry       A0: theNMRec (pointer)
      On exit        D0: result code (word)
```

_NMRemove removes the notification identified by nmReqPtr from the notification queue and returns one of the following result codes:

```
      Result codes   noErr                No error
                     qErr                 Not in queue
                     nmTypErr (-299)      qType field is not ORD(nmType)
```

## How to Call _NMInstall and _NMRemove

If you do not yet have glue for _NMInstall and _NMRemove, you can use the following from MPW Pascal:

```
      FUNCTION NMInstall (nmReqPtr: QElemPtr) : OSErr;
      INLINE $205F, $A05E, $3E80;

      FUNCTION NMRemove (nmReqPtr: QElemPtr) : OSErr;
      INLINE $205F, $A05F, $3E80;
```

Also note that qType must be set to ORD(nmType), which is 8.

The following short code segments demonstrate the use of the Notification Manager in MPW C 3.0.

```
      #include <OSUtils.h>
      #include <Notification.h>
```

```
struct NMRec   myNote;              /* declare your NMRec */
Handle         ManDoneS;            /* declare a handle for the sound */
OSErr          err;                 /* declare for err handling */

myNote.qType = nmType;              /* queue type -- nmType = 8 */
myNote.nmMark = 1;                  /* get mark in Apple menu */
myNote.nmSIcon = nil;               /* no flashing Icon */

/* get the sound you want out of your resources */
ManDoneS = GetResource('snd ', soundID);

myNote.nmSound = ManDoneS;          /* set the sound to be played
myNote.nmStr = nil;                 /* no alert box */
myNote.nmResp = nil;                /* no response procedure */
myNote.nmRefCon = nil;              /* set to nil since I don't need my A5*/
```

Before calling `_NMInstall`, you need to see if your application is running in the background. If your application is in the foreground, you really do not need to notify the user, but if your application is in the background, you should make the following call to install the notification event:

```
err = NMInstall ((QElemPtr) &myNote);
```

Before continuing, you should handle any errors. If your application is running in the background, you should wait until it switches to the foreground before proceeding with anything else. The example below is one method for detecting the switch to the foreground; it will continue waiting unless you get an event which is an `app4Evt` and a resume event.

```
while (!WaitNextEvent(everyEvent, &myEvent, yieldTime, nil)
        || myEvent.what != app4Evt
        || (myEvent.message << 31) == 0);

err = NMRemove ((QElemPtr) &myNote);
```

Once again, you should be sure to handle any errors.