Notification

by Ewout Walraven

File Notification is a form of interprocess communication available under Release 2.0. An application can ask a file system (like the RAM disk handler RAM:, df0:, df1:...) that supports notification to inform it whenever changes are made to a specific file or directory, making it easy for the application to react to such changes. The V37 ROM file system and the V37 and V36 RAM disk handler support file notification.

Under Release 2.0, the preferences control program, *IPrefs*, sets up notification on most of the preferences files in *ENV:sys*. If the user alters any of these files (which he/she normally does with a preferences editor), the system will notify *IPrefs* about the change. *IPrefs* will react to this notification by attempting to alter the user's environment to reflect the preference change. For example, if the user opens the *ScreenMode* preferences editor and alters the Workbench environment so that the Workbench screen should be a Hires NTSC screen, ScreenMode writes a file called *Screenmode.prefs* to the *ENV:sys* directory which happens to be in RAM:. Because *IPrefs* has set up notification on this file, the RAM disk file system will notify *IPrefs* of the change, *IPrefs* will read in the *Screenmode.prefs* file and will try to reset the Workbench screen so it is in Hires NTSC mode.

Notification allows very different applications to share common data files without knowing anything about each other. This has many possible uses in the Amiga's single user, multitasking environment. One possible use for notification is in a desktop publishing (DTP) package. The user can open the DTP package to layout a group of ILBMs, some structured drawings, and word processed text. When the user loads each of these, the DTP package sets up notification on each of their corresponding files. If the user loads an appropriate editor and changes any of the files on which the DTP package has set up notification, the DTP package will receive notification of these changes and can automatically re-import these files into the current DTP document without the user having to intervene. Another possible use for notification might be in a *make* utility. A *make* program for a compiler could set up notification on a set of source code and object files. If any of those files change, the *make* program will recompile and link the program, without the programmer having to intervene.

Setting up file notification on a file is easy. The StartNotify() function from *dos.library* starts notification on a file or directory:

```
BOOL StartNotify( struct NotifyRequest *notify );
```

StartNotify() returns DOSTRUE if the call is successful, or it returns DOSFALSE (for example, when the file's file system does not support notification). This function takes a pointer to an initialized NotifyRequest structure as its only argument (as defined in *<dos/notify.h>*):

```
struct NotifyRequest {
    UBYTE *nr_Name;     /* File/directory name for which you want notification */
    UBYTE *nr_FullName; /* Used by DOS. Do not use */
    ULONG nr_UserData;  /* For applications use */
    ULONG nr_Flags;     /* Flags indicating Signal or Message notification */

    union {
        /* Used for Message notification */
        struct {
            struct MsgPort *nr_Port;    /* Message port to receive messages on */
        } nr_Msg;
        /* Used for Signal notification */
        struct {
            struct Task *nr_Task;       /* The task to signal */
            UBYTE nr_SignalNum;         /* The signal number to use. */
            UBYTE nr_pad[3];
        } nr_Signal;
    } nr_stuff;

    ULONG nr_Reserved[4];          /* leave 0 for now */

    /* Used internally by handlers */
    ULONG nr_MsgCount;             /* number of outstanding messages */
    struct MsgPort *nr_Handler;    /* handler to send to (for EndNotify) */
};
```

This structure must not be altered by the application while notification is in effect!

The nr_Name field contains a pointer to the name of the file on which to set up notification. Currently, nr_Name has to be a file name and path containing a logical device name (for example df0:, work:, fonts:). The nr_FullName field is for the private use of the file system. Any other use of it is strictly prohibited. The nr_UserData field is available for an applications private use.

The nr_Flags field tells the file system which type of notification to set up, *message* or *signal*. When the file system uses message notification, it notifies an application by sending an Exec message. An application asks a file handler to notify it via an Exec message by setting the `NRF_SEND_MESSAGE` flag in nr_Flags. When the file system uses signal notification, it sets an Exec signal to notify an application. An application receives notification via a signal by setting the `NRF_SEND_SIGNAL` flag.

The nr_Flags field has two other flags, `NRF_WAIT_REPLY` and `NRF_NOTIFY_INITIAL`. The `NRF_WAIT_REPLY` tells the file handler not to send notification messages about a specific file/directory to an application if the application has not replied to a previous notification message about that specific file. This flag only applies to message notification. The `NRF_NOTIFY_INITIAL` flag tells the file handler to notify the application if the file exists when it sets up notification on the file. The flags for the nr_Flags field are defined in *<dos/notify.h>*.

The layout of the rest of the NotifyRequest structure depends on the type of notification. If the application is using message notification, it must supply the handler with a message port to send the notification messages. The NotifyRequest.nr_stuff.nr_Msg.nr_Port field contain the pointer to the message port that will receive the message notifications. If the application is using a signal for notification, it must supply a pointer to the task to signal and the number (not bit!) of the signal. In this case, the NotifyRequest.nr_stuff.nr_Signal.nr_Task field should contain the appropriate task pointer and the NotifyRequest.nr_stuff.nr_Signal.nr_SignalNum field should contain the signal number.

When a file handler uses message notification, it will send a NotifyMessage:

```
struct NotifyMessage {
    struct Message nm_ExecMessage;
    ULONG  nm_Class;              /* Class, will be NOTIFY_CLASS */
    UWORD  nm_Code;               /* Code, will be NOTIFY_CODE */
    struct NotifyRequest *nm_NReq; /* Pointer to the NotifyRequest you supplied */
    ULONG  nm_DoNotTouch;         /* private */
    ULONG  nm_DoNotTouch2;        /* private */
};
```

Message notification is especially useful if you are monitoring more than one file. It quickly enables you to find out which file/directory caused this message by either comparing the NotifyRequest structure returned in nm_NReq with the one you sent in the StartNotify() function, or by reading the NotifyRequest's nr_UserData field. Because the NotifyMessage's nm_Class and nm_Code fields contain values that distinguish it from other types of messages, you can use an already allocated message port (from a window for example) to receive notification messages.

To end notification on a file, use the *dos.library* function EndNotify():

```
void EndNotify( struct NotifyRequest *notify );
```

An application must call this function for each of its successful StartNotify() calls. This function takes one parameter, a pointer to the NotifyRequest structure that the application used to initiate the notification. In the case of message notification, EndNotify() will remove all pending notify messages from your message port. After calling this function, it is safe for the application to change or free the NotifyRequest structure. The application may also remove the message port or free the signal bit.

A file handler should send notification when it receives any of the following packets (from *<dos/dosextens.h>*) about the notification file or directory:

```
ACTION_RENAME_OBJECT
ACTION_RENAME_DISK
ACTION_CREATE_DIR
ACTION_DELETE_OBJECT

ACTION_WRITE
ACTION_FINDUPDATE
ACTION_FINDOUTPUT
ACTION_SET_FILE_SIZE

ACTION_SET_DATE
```

The first four packets will cause notification immediately. The second four packets will cause notification when the notification file is closed. The last packet, `ACTION_SET_DATE`, should cause notification immediately, but due to a bug in the V37 ROM file system, only the RAM disk's file handler (RAM:) will send notification.

Notice that some of the packets that trigger a notification are sent by a process when it is trying to create a new file or directory. A file system that supports notification should be able to set up notification on a file or directory that does not currently exist. A file system should send notification when it creates that file or directory.

Note however that, although notification on directories is part of the OS in release 2.0, it does not work correctly. In the ROM file system, *directory notification only works if that directory exists when notification is set up*. If your application tries to set up notification on a ROM file system directory before the directory exists, *your application will* ***never*** *receive notification about that directory*. If notification is set up for a directory in RAM:, you will only be informed when that directory is created or deleted and when files are created in that directory, not when files are changed or deleted.

When implementing notification in your application, there are several things to remember. Not every file system supports notification, in particular, most network file systems will not support notification. For this reason, no application should require notification to function.

At the end of this article are two examples for file notification. *SignalNotification.c* implements signal notification on a single target. *MessageNotification.c* shows how to start message notification on multiple targets. Note that these examples are not linked with startup code (like *c.o*). Although these examples do set up SysBase and DosBase to gain access to *exec.library* and *dos.library*, the examples do not handle the startup message (WBenchMsg) that Workbench sends when it launches an application, *so do not run these examples from Workbench*.