Handler Maintenance and Control

A number of packets are defined to give an application some control over a file system:

```
ACTION_DIE              5      <sendpkt only>
RES1:   BOOL    DOSTRUE
```

As its name implies, the `ACTION_DIE` packet tells a handler to quit. All new handlers are expected to implement this packet. Because of outstanding locks and the fact that the handler address is returned by the DeviceProc() routine, it is unlikely that the handler can disappear completely, but instead will have to release as many resources as possible and simply return an error on all packets sent to it.

In the future, the system may be able to determine if there are any outstanding DeviceProc() references to a handler, and therefore make it is safe to shut down completely.

```
ACTION_FLUSH            27     <sendpkt only>
RES1:   BOOL    DOSTRUE
```

This action causes the file system to flush out all buffers to disk *before* returning this packet. If any writes are pending, they must be processed before responding to this packet. This packet allows an application to make sure that the data that is supposed to be on the disk is actually written to the disk instead of waiting in a buffer.

```
ACTION_MORE_CACHE       18      AddBuffers(...) in 2.0
ARG1:   LONG    Number of buffers to add

RES1:   BOOL    DOSTRUE (-1L)
RES2:   LONG    New total number of buffers
```

This action allows an application to change the number of internal buffers used by the file system for caching. Note that a positive number increases the number of buffers while a negative number decreases the number of buffers. In all cases, the number of current buffers should be returned in RES2. This allows an application to inquire the number of buffers by sending in a value of 0 (resulting in no change). Note that the OFS and FFS in 1.3 do not accept a negative number of buffers.

Note that there is a bug in the ROM file system in both Release 2.04 and Release 3.0 that jumbles its return values for this packet. The file system erroneously returns the new number of buffers in RES1 instead of RES2 (it returns a failure code in RES2). To work around this bug when using this packet, test RES1 to see if it is DOSTRUE (-1L). If it is, look at RES2 for the number of buffers, otherwise RES1 should contain the new total number of buffers.

```
ACTION_INHIBIT            31     Inhibit(...) in 2.0
ARG1:   BOOL    DOSTRUE = inhibit,      DOSFALSE = uninhibit

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
```

This action is probably one of the most dangerous that a file system has to handle. When inhibited (ARG1 = DOSTRUE), the file system must not access any underlying media and return an error code on all attempts to access the device. Once uninhibited (ARG1 = DOSFALSE), the file system must *assume* that the medium has been changed. The file system must flush the buffers before the ACTION_INHIBIT , popping up a requester demanding that the user put back the current disk, if necessary. The handler may choose to reject an inhibit request if any objects are open for writing.

Although it's not required, a handler should nest inhibits. Prior to 2.0, the system handlers did not keep a nesting count and were subject to some obscure race conditions. The 2.0 ROM filing system introduced a nesting count.

```
ACTION_WRITE_PROTECT     1023    <sendpkt only>
ARG1:   BOOL    DOSTRUE/DOSFALSE (write protect/un-write protect)
ARG2:   LONG    32 Bit pass key

RES1:   BOOL    DOSTRUE/DOSFALSE
```

This is a new packet defined for the Fast File System. This packet allows an application to change the write protect flag of a disk (if possible - applications cannot write to floppies that have their write-protect tabs set). This packet is primarily intended to allow write-protecting non-removable media such as hard disks. The value in ARG1 toggles the write status. The 32-bit passkey allows a program to prevent other programs from unwrite-protecting a disk. To unlock a disk, ARG2 must match the passkey of the packet that locked the disk, unless the disk was locked with a passkey of 0. In this case, no passkey is necessary to unlock the disk.

2.0 only

```
ACTION_IS_FILESYSTEM     1027    IsFileSystem(devname)

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

Through this function, a handler can indicates whether or not it is a file system (whether or not it can support separate files for storing information). Programs will assume a handler can create multiple, distinct files through calls to Open() if the handler returns this packet with a DOSTRUE value. A handler does *not* need to support directories and subdirectories in order to qualify as a file system. It does have to support the Examine()/ExNext() calls.

Note that the AmigaDOS routine IsFileSystem() will attempt to use Lock(":",SHARED_ACCESS) if this packet returns ERROR_ACTION_NOT_KNOWN.

Handler Internal

There are several actions that are generally used by handlers to allow messages returning from requested services (typically an Exec device) to look like incoming request packets. This allows the handler to request an asynchronous operation but be notified of the completion. For example, a handler sends the *serial.device* a request for a read, but instead of sending a plain IO request, it sends a DOS packet disguised as an IO request. The *serial.device* treats the packet like a normal IO request, returning it to the handler when it is finished. When the handler gets back its disguised DOS packet, it knows that the read has completed.

ACTION_NIL 0 <internal>

Although not specifically an action, many returns look like this value because the action field has not been filled in.

ACTION_READ_RETURN 1001 <internal>

Generally used to indicate the completion of an asynchronous read request.

ACTION_WRITE_RETURN 1002 <internal>

Generally used to indicate the completion of an asynchronous write request.

ACTION_TIMER 30 <internal>

Used to indicate the passage of a time interval. Many handlers have a steady stream of ACTION_TIMER packets so that they can schedule house keeping and flush buffers when no activity has occurred for a given time interval.

Obsolete Packets

There are several packet types that are documented within the system include files that are obsolete. A file system is not expected to handle these packets and any program which sends these packets can not expect them to work:

ACTION_DISK_CHANGE 33 <Obsolete>

ACTION_DISK_TYPE 32 <Obsolete>

ACTION_EVENT 6 <Obsolete>

ACTION_GET_BLOCK 2 <Obsolete>

ACTION_SET_MAP 4 <Obsolete>

Of particular note here is ACTION_DISK_CHANGE. The *DiskChange* command uses the ACTION_INHIBIT packet to accomplish its task.

Console Only Packets

The remaining packets are only used for console handlers and do not need to be implemented by a file system.

```
ACTION_SCREEN_MODE       994    SetMode() in 2.0
ARG1:   LONG    Mode (zero or one)

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

Switch the console to and from RAW mode. An ARG1 of one indicates the unprocessed, raw mode while an ARG1 of zero indicates the processed, ‘‘cooked’’ mode.

2.0 only

```
ACTION_CHANGE_SIGNAL     995    <sendpkt only>
ARG1:   LONG    The fh_Arg1 of the console file handle
ARG2:   APTR    MsgPort of the process to signal
ARG3:   LONG    Reserved, currently this must be zero

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

This packet redirects what process the console handler signals when the user hits Control-C, Control-D, Control-E, or Control-F. Normally the process that opened the file handle receives the break signal.

```
ACTION_WAIT_CHAR         20     WaitForChar()
ARG1:   ULONG   Timeout in microseconds

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

Performs a timed read of a character. The WaitForChar() function uses this packet.

```
ACTION_DISK_INFO         25      <sendpkt only>
ARG1:   BPTR    Pointer to an InfoData structure to fill in

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
```

The `ACTION_DISK_INFO` packet has a special meaning for console style handlers. When presented with this packet, a console style handler should return a pointer to the window associated with the open handle in the InfoData structure’s id_VolumeNode field (the InfoData structure is defined in *<dos/dos.h>*). Note that some consoles can return a `NULL` Window pointer (for example, an AUTO CON: or a AUX: console). The Amiga’s standard console handler, CON:, also returns a pointer to the console handler’s IO request in the id_InUse field. In some cases, the IO request’s io_Unit field (which normally point to a ConUnit structure) will be NULL. See also the `ACTION_DISK_INFO` packet in the ‘‘Volume Manipulation/Information’’ section.