Basic Input/Output

The Basic Input/Output actions are supported by both handlers and file systems. In this way, the application can get a stream level access to both devices and files. One difference that arises between the two is that a handler will not necessarily support an ACTION_SEEK while it is generally expected for a file system to do so.

These actions work based on a FileHandle which is filled in by one of the three forms of opens:

```
ACTION_FINDINPUT         1005    Open(..., MODE_OLDFILE)
ACTION_FINDOUTPUT        1006    Open(..., MODE_NEWFILE)
ACTION_FINDUPDATE        1004    Open(..., MODE_READWRITE)
ARG1:  BPTR FileHandle to fill in
ARG2:  LOCK Lock on directory that ARG3 is relative to
ARG3:  BSTR Name of file to be opened (relative to ARG1)

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

All three actions use the lock (ARG2) as a base directory location from which to open the file. If this lock is NULL, then the file name (ARG3) is relative to the root of the current volume. Because of this, file names are not limited to a single file name but instead can include a volume name (followed by a colon) and multiple slashes allowing the file system to fully resolve the name. This eliminates the need for AmigaDOS or the application to parse names before sending them to the file system. Note that the lock in ARG2 must be associated with the file system in question. It is illegal to use a lock from another file system.

The calling program owns the file handle (ARG1). The program must initialize the file handle before trying to open anything (in the case of a call to Open(), AmigaDOS allocates the file handle automatically and then frees it in Close()). All fields must be zero except the fh_Pos and fh_End fields which should be set to -1. The Open() function fills in the fh_Type field with a pointer to the MsgPort of the handler process. Lastly, the handler must initialize fh_Arg1 with something that allows the handler to uniquely locate the object being opened (normally a file). This value is implementation specific. This field is passed to the READ/WRITE/SEEK/ END/TRUNCATE operations and not the file handle itself.

FINDINPUT and FINDUPDATE are similar in that they only succeed if the file already exists. FINDINPUT will open with a shared lock while FINDUPDATE will open it with a shared lock but if the file doesn't exist, FINDUPDATE will create the file. FINDOUTPUT will always open the file (deleting any existing one) with an exclusive lock.

```
ACTION_READ              'R'     Read(...)
ARG1:  ARG1 fh_Arg1 field of the opened FileHandle
ARG2:  APTR Buffer to put data into
ARG3:  LONG Number of bytes to read

RES1:  LONG Number of bytes read.
        0 indicates EOF.
       -1 indicates ERROR
RES2:  CODE Failure code if RES1 is -1
```

This action extracts data from the file (or input channel) at the current position. If fewer bytes remain in the file than requested, only those bytes remaining will be returned with the number of bytes stored in RES1. The handler indicates an error is indicated by placing a -1 in RES1 and the error code in RES2. If the read fails, the current file position remains unchanged. Note that a handler may return a smaller number of bytes than requested, even if not at the end of a file. This happens with interactive type file handles which may return one line at a time as the user hits return, for example the console handler, CON:.

```
ACTION_WRITE              'W'     Write(...)
ARG1:  ARG1 fh_Arg1 field of the opened file handle
ARG2:  APTR Buffer to write to the file handle
ARG3:  LONG Number of bytes to write

RES1:  LONG Number of bytes written.
RES2:  CODE Failure code if RES1 not the same as ARG3
```

This action copies data into the file (or output channel) at the current position. The file is automatically extended if the write passes the end of the file. The handler indicates failure by returning a byte count in RES1 that differs from the number of bytes requested in ARG3. In the case of a failure, the handler does not update the current file position (although the file may have been extended and some data overwritten) so that an application can safely retry the operation.

```
ACTION_SEEK               1008    Seek(...)
ARG1:  ARG1    fh_Arg1 field of the opened FileHandle
ARG2:  LONG    New Position
ARG3:  LONG    Mode:   OFFSET_BEGINNING,OFFSET_END, or  OFFSET_CURRENT

RES1:  LONG    Old Position.   -1 indicates an error
RES2:  CODE    Failure code if RES1 = -1
```

This packet sets the current file position. The new position (ARG2) is relative to either the beginning of the file (`OFFSET_BEGINNING`), the end of the file (`OFFSET_END`), or the current file position (`OFFSET_CURRENT`), depending on the mode set in ARG3. Note that ARG2 can be negative. The handler returns the previous file position in RES1. Any attempt to seek past the end of the file will result in an error and will leave the current file position in an unknown location.

```
ACTION_END                1007    Close(...)
ARG1:   ARG1    fh_Arg1 field of the opened FileHandle

RES1:   LONG    DOSTRUE
```

This packet closes an open file handle. This function generally returns a `DOSTRUE` as there is little the application can do to recover from a file closing failure. If an error is returned under 2.0, DOS will not deallocate the file handle. Under 1.3, it does not check the result.