This chapter describes the AppleTalk Filing Protocol (AFP) that allows a workstation on an AppleTalk network to access and manipulate files on an AFP file server, such as an AppleShare server.

Because you can use the native file system to access an AFP server from a workstation, in most cases you should not need to use AFP directly. For example, few application developers use AFP to access an AppleShare file server because the existing File Manager commands perform most of the functions needed to access and manipulate files on an AppleShare server.

However, if you want to provide functions that are not implemented by the native file system commands or you want to manipulate files on an AFP server other than an AppleShare server, your application can use the AFP programming interface to directly access AFP to send commands to the server. For example, you can use AFP to list the contents of a directory when you need to obtain ProDOS information. You can also use AFP to retrieve or set parameters for a specific file when ProDOS is used.

This chapter describes the programming interface to the workstation portion of AFP only. It does not describe how to implement an AFP server. For information on how to implement an AFP server, see *Inside AppleTalk*, second edition.

Because AFP is not widely used by application program developers, this chapter provides only the AFP basics. This chapter includes "About" and "Reference" sections. It does not include a "Using" section, as do most of the other chapters in this book. This chapter is included in this book to complete the coverage of the AppleTalk protocol stack.

If you decide to use AFP, it is important to note that to implement an AFP command, you need information in addition to the information that this chapter provides. *Inside AppleTalk*, second edition, and the *AppleShare 3.0 Developer's Kit* version 3.0, provide information describing the AFP commands and the command block data structure required for each command. The *AppleShare 3.0 Developer's Kit* includes extensions to AFP not described in *Inside AppleTalk*.

AFP is built on top of the AppleTalk Session Protocol (ASP) and uses the services of ASP. To use AFP, you should also be familiar with ASP, which is described in the chapter "AppleTalk Session Protocol (ASP)" in this book. For an overview of AFP and how it fits within the AppleTalk protocol stack, read the chapter "Introduction to AppleTalk," which is also in this book.

About AFP

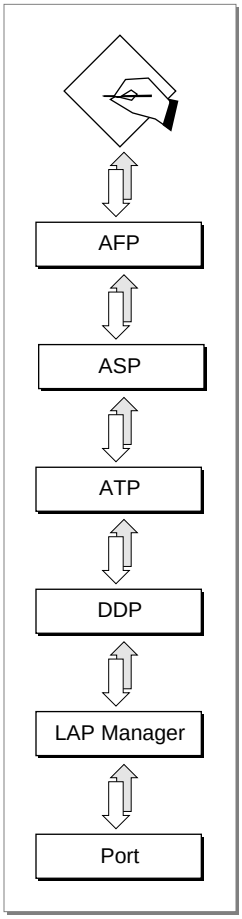
AFP is a remote filing system protocol that provides a workstation on an AppleTalk network with access to a server that is implemented according to the AFP file system structure. AFP also includes user authentication support and an access control mechanism that supports volume-level and folder-level access rights. AppleShare is the AFP file server that is implemented on Macintosh computers.

Through the native file system and AFP, your application running on one node can manipulate files on another node using the same file system commands on the remote node that it uses to manipulate files on its own node. You can use AFP commands to

- obtain and modify information about the file server and other parts of the file system structure
- create and delete files and directories
- read files or write to them
- retrieve and store information within individual files

AFP is implemented by the .XPP driver. The .XPP driver maps an AFP function call from the client workstation into one or more ASP function calls. Figure 9-1 shows AFP and its underlying protocols.

Figure 9-1 AFP and its underlying protocols



The Pascal programming interface to AFP on the workstation consists of a single function. You use this function to pass to the .XPP driver the command code and parameters for an AFP command. There are four categories of AFP commands: general, login, read, and write. Each of these categories requires a specific format of the XPP parameter block that is used for the AFP function. The next section describes these categories, the commands they include, and the XPP parameter block formats for each category.

Please read this note before you continue

Because the native file system commands implement most of the functions that you need to access an AFP server, in most cases you will not need to use AFP directly. For this reason, this chapter does not include a "Using" section, as do most of the other chapters in this book. If the native file system implements the function that you need, you should use the file system command. If you want to implement a function that is not part of the native file system, you can use AFP directly. In this case, you should continue to read this chapter. ◆

AFP Reference

This section describes the data structures and the function that are specific to the AppleTalk Filing Protocol (AFP).

The "Data Structures" section shows the Pascal data structures for the AFP command block record and the XPP parameter block.

The AFP programming interface consists of a single function, `AFPCommand`, which allows you to call AFP and specify from within a command block a particular command and its parameters to send across the session to the server.

Data Structures

This section describes the data structures that you use to provide information to the AppleTalk Filing Protocol (AFP).

You use an AFP command block record for the AFP read or AFP write format of the `AFPCommand` function.

You use the XPP parameter block as a parameter to the `AFPCommand` function.

AFP Command Block Record

An AFP command block record of type `AFPCommandBlock` defines the structure of the command block that you use to send either a read (`afpRead`) or write (`afpWrite`) command to the server. A ***command block*** is a data structure that is used to specify an AFP command and its parameters, which the .XPP driver sends to an AFP server to be executed. The XPP parameter block for the `AFPCommand` function contains a pointer to

the command block. The read and write commands use different fields of this record. You pass a pointer to the AFP command block record as a field value of the XPP parameter block. The command block record fields are defined in the section describing the command that uses them.

```
AFPCommandBlock =
PACKED RECORD
   cmdByte:             Byte;    {AFP command code}
   startEndFlag:        Byte;    {start/end flag; for the read }
                                 { command, identifies offset }
                                 { relative to fork}
   forkRefNum:          Integer; {fork reference number}
   rwOffset:            LongInt; {offset within fork to begin }
                                 { reading or writing}
   reqCount:            LongInt; {on input, requested size of }
                                 { data; on return, size of data }
                                 { actually read or written}
   newLineFlag:         Byte;    {new line flag}
   newLineChar:         Char;    {new line character}
END;
```

XPP Parameter Block

The `AFPCommand` function, which has four formats, requires a pointer to an XPP parameter block of type `XPPParamBlock`. Because the .XPP driver maps the AFP commands that you specify to ASP commands, the various `AFPCommand` function formats use some of the XPP parameter block fields defined within variant records for ASP functions.

The first four fields of the XPP parameter block, `qLink`, `qType`, `ioTrap`, and `ioCmdAddr`, which are the same for all four formats of the `AFPCommand`, are used internally by the Device Manager.

You must specify the .XPP driver reference number as the input value of the `ioRefNum` field; AFP does not fill in this value. You can use the Device Manager's `OpenDriver` function to obtain the .XPP driver reference number.

The XPP parameter block that follows is defined as the maximum size required for any format of the `AFPCommand` function. Various formats use different size parameter blocks. You can abbreviate the parameter block appropriately for any `AFPCommand` format.

This section defines the parameter block fields that are common to all AFP functions. It does not define reserved fields, which are used either internally by the .XPP driver or not at all. The fields that are used by a particular format are defined in the section that describes that format.

```
XPPPrmBlkType = (XPPPrmBlk...);
XPPSubPrmType = (ASPOpenPrm,ASPSubPrm);
XPPEndPrmType = (AFPLoginPrm,ASPEndPrm);
XPPParmBlkPtr = ^XPPParamBlock;
XPPParamBlock =
PACKED RECORD
   qLink:               QElemPtr;         {reserved}
   qType:               Integer;          {reserved}
   ioTrap:              Integer;          {reserved}
   ioCmdAddr:           Ptr;              {reserved}
   ioCompletion:        ProcPtr;          {completion routine}
   ioResult:            OSErr;            {result code}
   cmdResult:           LongInt;          {command result (ATP user bytes)}
   ioVRefNum:           Integer;          {reserved}
   ioRefNum:            Integer;          {driver reference number}
   csCode:              Integer;          {call command code}
CASE XPPPrmBlkType OF
   XPPPrmBlk:
      (sessRefnum:      Integer;          {offset to session refnum}
      aspTimeout:       Byte;             {timeout for ATP}
      aspRetry:         Byte;             {retry count for ATP}
CASE XPPSubPrmType OF
   ASPOpenPrm:
      (serverAddr:   AddrBlock;           {server address block}
      scbPointer:    Ptr;                 {SCB pointer}
      attnRoutine:   Ptr);                {attention routine pointer}
   ASPSubPrm:
      (cbSize:       Integer;             {command block size}
      cbPtr:         Ptr;                 {command block pointer}
      rbSize:        Integer;             {reply buffer size}
      rbPtr:         Ptr;                 {reply buffer pointer}
CASE XPPEndPrmType OF
   AFPLoginPrm:
      (afpAddrBlock: AddrBlock;           {address block in AFP login}
      afPSCBPtr:     Ptr;                 {SCB pointer in AFP login}
      afpAttnRoutine:Ptr);                {attn routine pointer in AFP login}
   ASPEndPrm:
      (wdSize:       Integer;             {write data size}
      wdPtr:         Ptr;                 {write data pointer}
      ccbStart:      ARRAY[0..295] OF Byte)));
                                          {beginning of command control }
                                          { block}
END;
XPPParmBlkPtr = ^XPPParamBlock;
```

Field descriptions

`ioCompletion` A pointer to a completion routine that you can provide. When you execute a function asynchronously, the .XPP driver calls your completion routine when it completes execution of the function if you specify a pointer to the routine as the value of this field. Specify `NIL` for this field if you do not wish to provide a completion routine. If you execute a function synchronously, the .XPP driver ignores the `ioCompletion` field. For information about completion routines, see the chapter "Introduction to AppleTalk" in this book.

`ioResult` The result of the function. When you execute the function asynchronously, the function sets this field to 1 and returns a function result of `noErr` as soon as the function begins execution. When the function completes execution, it sets the `ioResult` field to the actual result code.

`ioRefNum` The driver reference number for the .XPP driver. The Device Manager's `OpenDriver` function that you use to open the .XPP driver returns the driver reference number in the `refnum` field. You must supply this value. You can call this function to obtain the .XPP driver's reference number even if the .XPP driver is already open. The MPW interface does not fill in this value. For information on opening the .XPP driver, see the chapter "AppleTalk Utilities" in this book. For information on the `OpenDriver` function, see the chapter "Device Manager" in *Inside Macintosh: Devices.*

`csCode` The .XPP driver command code for the function. For the `AFPCommand` function, the value of this field is always `afpCall`. The MPW interface fills in this field.

Routines

The programming interface to AFP is different in form from the programming interfaces to the other AppleTalk protocols described in this book. For AFP, the programming interface consists of a single function, the `AFPCommand` function, which allows you to call AFP and pass it the command code for a particular AFP command. There are four categories or types of commands that you can send to a server: general, login, write, and read. To use the commands that form these categories, in addition to this chapter, you must also refer to the books *Inside AppleTalk*, second edition, and *AppleShare 3.0 Developer's Kit* version 3.0.

The `AFPCommand` function requires as a parameter a pointer to an XPP parameter block. This function uses a different parameter block format for each category. You do not specify the command code as a parameter block field value, as you might expect. Instead, as the value of a parameter block field you specify a pointer to a command buffer. You use the command buffer to specify the command code of the AFP command to be sent to the server.

Although the `AFPCommand` function syntax is the same for all four formats, the fields of the XPP parameter block that are used for each format differ. The `AFPCommand` function is defined as follows:

```
FUNCTION AFPCommand (thePBptr: XPPParmBlkPtr;
                     async: Boolean): OSErr;
```

`thePBptr` A pointer to the XPP parameter block format for a particular group of AFP commands.

`async` A Boolean that specifies whether the function is to execute synchronously or asynchronously. Set the `async` parameter to `TRUE` to execute the function asynchronously.

This section describes the XPP parameter block format for each category of commands. An arrow preceding a parameter block field indicates whether the field's value is an input parameter, an output parameter, or both:

Arrow	Meaning
→	Input
←	Output
↔	Both

Within the parameter block, you specify a pointer to a command block, the first byte of which contains the command code of the command to be sent to the server. The range of command codes is 0 through 255, inclusive, although AppleTalk does not currently implement all command codes and some command codes are invalid. Table 9-1 shows the AFP command codes that are implemented in AppleTalk. This table shows the AFP command code constant, the numeric value, and a description of the command.

Note
The following six constants may not be defined in the header files: `afpGetSrvrMsg`, `afpCreateID`, `afpDeleteID`, `afpResolveID`, `afpExchangeFiles`, and `afpCatSearch`. If you use the commands that these constants identify, you must either specify the numeric values for the commands or declare the constants in your application. ◆

Table 9-1 AFP command codes

AFP command constant	Command code	Action
`afpByteRangeLock`	1	Locks or unlocks a specified range of bytes within an open fork.
`afpVolClose`	2	Informs the server that the workstation no longer needs the volume.
`afpDirClose`	3	Closes a directory and invalidates its directory identifier.

continued

Table 9-1 AFP command codes (continued)

AFP command constant	Command code	Action
`afpForkClose`	4	Closes a fork that was opened by `afpOpenFork`.
`afpCopyFile`	5	Copies a file from one location to another on the same file server.
`afpDirCreate`	6	Creates a new directory.
`afpFileCreate`	7	Creates a new file.
`afpDelete`	8	Deletes a file or directory.
`afpEnumerate`	9	Lists the contents of a directory.
`afpFlush`	10	Writes to a disk any volume data that has been modified.
`afpForkFlush`	11	Writes to a disk any data buffered from previous `afpWrite` calls.
`afpGetForkParms`	14	Retrieves parameters for a file associated with a particular open fork.
`afpGetSInfo`	15	Obtains a block of descriptive information from the server, without requiring an open session. Use the `ASPGetStatus` function instead of this command code. See the chapter "AppleTalk Session Protocol (ASP)" in this book for information on `ASPGetStatus`. Making an `afpGetSInfo` call using the `AFPCommand` results in an error.
`afpGetSParms`	16	Retrieves server parameters.
`afpGetVolParms`	17	Retrieves parameters for a particular volume.
`afpLogin`	18	Establishes an AFP session with a server.
`afpContLogin`	19	Continues the login and user authentication process started by the `afpLogin` command.
`afpLogout`	20	Terminates a session with a server.
`afpMapID`	21	Maps a user ID to a user name, or a group ID to a group name.
`afpMapName`	22	Maps a user name to a user ID, or a group name to a group ID.
`afpMove`	23	Moves a directory or file to another location on the same volume.
`afpOpenVol`	24	Makes a volume available to the workstation.
`afpOpenDir`	25	Opens a directory on a variable directory ID volume and returns its directory ID.

continued

Table 9-1 AFP command codes (continued)

AFP command constant	Command code	Action
`afpOpenFork`	26	Opens the data or resource fork of an existing file to read from it or write to it.
`afpRead`	27	Reads a block of data from an open fork.
`afpRename`	28	Renames a directory or file.
`afpSetDirParms`	29	Sets parameters for a specified directory.
`afpSetFileParms`	30	Sets parameters for a specified file.
`afpSetForkParms`	31	Sets the fork length for a specified open fork.
`afpSetVolParms`	32	Sets the backup date for a specified volume.
`afpWrite`	33	Writes a block of data to an open fork.
`afpGetFlDrParms`	34	Retrieves parameters for either a file or a directory.
`afpSetFlDrParms`	35	Sets parameters for a file or directory.
`afpGetSrvrMsg`*	38	Gets a string message from the server, such as shutdown, user, and login messages.
`afpCreateID`*	39	Creates a unique file ID for a specified file.
`afpDeleteID`*	40	Invalidates all instances of a specified file ID.
`afpResolveID`*	41	Returns parameters for the file referred to by the specified file ID.
`afpExchangeFiles`*	42	Preserves an existing file ID when an application performs a “Save” or “Save As” operation.
`afpCatSearch`*	43	Allows an application to efficiently search an entire volume for files that match specified criteria.
`afpDTOpen`	48	Opens the Desktop database on a particular volume.
`afpDTClose`	49	Informs the server that the workstation no longer needs the volume’s Desktop database.
`afpGetIcon`	51	Retrieves an icon from the volume’s Desktop database.
`afpGtIcnInfo`	52	Retrieves icon information from the volume’s Desktop database.
`afpAddAPPL`	53	Adds an APPL mapping to the Desktop database.
`afpRmvAPPL`	54	Removes an APPL mapping from the volume’s Desktop database.

continued

Table 9-1 AFP command codes (continued)

AFP command constant	Command code	Action
`afpGetAPPL`	55	Retrieves an APPL mapping from the volume's Desktop database.
`afpAddCmt`	56	Adds a comment for a file or directory to the volume's Desktop database.
`afpRmvCmt`	57	Removes a comment from the volume's Desktop database.
`afpGetCmt`	58	Retrieves a comment associated with a specified file or directory from the volume's Desktop database.
`afpAddIcon`	192	Adds an icon bitmap to the volume's Desktop database.

* An asterisk (*) marks the constants that may not be defined in the header files. If you use them, you must first declare the constants in your application.

The command block buffer that you provide for each `AFPCommand` function contains the command code and the command parameters. The format for the command block differs for each command.

For a description of the commands and their required command block formats and parameters, see *Inside AppleTalk*, second edition, and the *AppleShare 3.0 Developer's Kit* version 3.0 as follows:

- For command codes 38 through 43, inclusive, see the *AppleShare 3.0 Developer's Kit* version 3.0.
- For all other AFP command codes, see *Inside AppleTalk*, second edition.

The .XPP driver implements most AFP commands by mapping the AFP command to an ASP function, without interpreting or verifying the data. The .XPP driver maps AFP commands to ASP functions according to the following conventions:

AFP commands are mapped to ASP functions, which use the services of ATP to transport data. The following two AFP command codes can send or receive more data than a single eight-packet ATP transaction will support.

- The `afpRead` command (27) can cause the server to return more data than fits in eight ATP response packets. (The `aspQuantumSize` parameter of the `ASPGetParms` function returns the maxi`mum` amount of data that you can receive from the server.) The `afpRead` command can return up to the number of bytes indicated in the command block's requested count (`reqCount`) field. The .XPP driver may issue multiple calls to ASP for this command mapping.
- The `afpWrite` command (33) can pass more data than fits in eight ATP response packets. The `afpWrite` command can pass up to the number of bytes indicated in the command block's requested count (`reqCount`) field. The .XPP driver may issue multiple calls to ASP for this command mapping.

Table 9-2 summarizes the mapping of AFP commands to ASP functions.

Table 9-2 Mapping of AFP commands to ASP functions

AFP command code	ASP function mapping
0	Invalid AFP command.
1–14, 16, 17, 21–32, 34–190	Mapped to `ASPUserCommand`.
15	Mapped to `ASPGetStatus`. Use `ASPGetStatus` instead of this command code. Making an `afpGetSInfo` call using the `AFPCommand` function results in an error.
18	Mapped to appropriate login dialog including `ASPOpenSession`.
19	Mapped to appropriate login dialog.
20	Mapped to `ASPCloseSession`.
33	Mapped to `ASPUserWrite`.
191	Mapped to `ASPUserCommand`. Reserved for developers; Apple Computer, Inc., will not use this command code.
192–253	Mapped to `ASPUserWrite`.
254	Mapped to `ASPUserWrite`. Reserved for developers; Apple Computer, Inc., will not use this command code.
255	Invalid AFP command.

Before you can call the `AFPCommand` function, you must open the .XPP driver. You can use the Device Manager's `OpenDriver` function to open the .XPP driver. You should not close the .XPP driver because other applications and processes may be using it. For more information on opening the .XPP driver, see the chapter "AppleTalk Utilities" in this book. The .MPP and .ATP drivers must be open before you open the .XPP driver.

The chapter "AppleTalk Utilities" also describes how to close the .XPP driver. However, in most circumstances, you should not close the .XPP driver because other applications and processes could be using the protocols implemented by the .XPP driver.

You must pass the .XPP driver reference number as a parameter to the `AFPCommand` function; the MPW interface does not fill in this value. The `OpenXPP` function that you use to open the .XPP driver returns the driver reference number in the `refnum` field. You can call this function to obtain the .XPP driver's reference number even if the .XPP driver is already open.

For all `AFPCommand` formats, the XPP parameter block includes a `CCBStart` field. The .XPP driver uses the memory at the end of the XPP parameter block defined as a `CCBStart` array as an internal command control block (CCB). To ensure that the function executes successfully, you can specify the maximum size for this array as indicated for the particular function that uses it.

AFP General Command Format

You use the general command format for the `AFPCommand` function to pass any of the AFP commands to the .XPP driver to be sent to the server except `afpLogin`, `afpRead`, and `afpWrite`.

```
FUNCTION AFPCommand (thePBptr: XPPParmBlkPtr;
                     async: Boolean): OSErr;
```

`thePBptr` A pointer to the XPP parameter block format for the AFP commands that use the AFP general command format.

`async` A Boolean that specifies whether the function is to be executed synchronously or asynchronously. Set the `async` parameter to `TRUE` to execute the function asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
←	`cmdResult`	`LongInt`	The AFP command result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `afpCall` for this function.
→	`sessRefnum`	`Integer`	The session reference number.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`cbSize`	`Integer`	The command buffer size.
→	`cbPtr`	`Ptr`	The command buffer.
↔	`rbSize`	`Integer`	The reply buffer size and reply size.
→	`rbPtr`	`Ptr`	A pointer to the reply buffer.
↔	`wdSize`	`Integer`	The write data size.
→	`wdPtr`	`Ptr`	A pointer to the write data.

Field descriptions

`cmdResult` Four bytes of data returned from the server indicating the result of the AFP command.

`sessRefnum` The session reference number, which is a unique number that the .XPP driver assigns to the session and returns in response to an `afpLogin` command.

`aspTimeout` The interval in seconds that the .XPP driver waits between retries of the AFP command.

`cbSize` The size in bytes of the block of data that contains the command and its parameters to be sent to the server across the session. The size of the command block must not exceed the value of `aspMaxCmdSize` that the `ASPGetParms` function returns. For information on the `ASPGetParms` function, see the chapter "AppleTalk Session Protocol (ASP)" in this book.

`cbPtr` A pointer to the beginning of the command block that contains the AFP command to be sent across the session to the server. (The `cbSize` field value specifies the command block size.) The first byte of the command block must contain the AFP command. The

following bytes contain the parameters for the command. See *Inside AppleTalk*, second edition, and the *AppleShare 3.0 Developer's Kit* version 3.0 for the definitions of the AFP commands and their command codes and parameters.

`rbSize` On input, the size in bytes of the reply buffer that is to hold the expected response to the AFP command. On return, the actual size of the reply to the AFP command that the .XPP driver returned in the buffer.

`rbPtr` A pointer to the reply buffer.

`wdSize` The size of the write data buffer that contains the data to be written to the server. This field's value is used only if the AFP command is one that the .XPP driver maps to the `ASPUserWrite` function.

`wdPtr` A pointer to the write-data buffer. This field's value is used only if the AFP command is one that the .XPP driver maps to the `ASPUserWrite` function.

DESCRIPTION

The general format of the `AFPCommand` function provides a way to pass an AFP command to the server end of an open session and receive a reply. After you open a session with an AFP file server using the login format of the `AFPCommand` function, you can send a sequence of AFP commands across the session to the server. You use the general format for the `AFPCommand` function to send all of the AFP commands to the server, except for `afpLogin`, `afpRead`, and `afpWrite`, which have their own `AFPCommand` formats. AFP delivers the commands in the same order in which you send them and returns replies to the commands in the reply buffer that you provide. The `cmdResult` field indicates the result of the command that was delivered to the server, not the function result.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function. For information on the `OpenDriver` function, see *Inside Macintosh: Devices*.

Any memory that you allocate for the parameter block, buffers, and command block belongs to the .XPP driver until the function completes execution, after which you can reuse or release the memory.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `AFPCommand` function from assembly language, call the `_Control` trap macro with a value of `afpCall` in the `csCode` field of the parameter block.

RESULT CODES

`aspBufTooSmall`	–1067	The command reply from the server is larger than the response buffer; ASP will fill the buffer and truncate the reply data
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspSessClosed`	–1072	The .XPP driver is in the process of closing the session
`aspSizeErr`	–1073	The size of the command block exceeds the maximum size of `aspMaxCmdSize`
`afpParmError`	–5019	The AFP command block size is equal to 0 (this error is also returned when the command block is equal to 0 or $FF [255] or `GetSrvrStatus` [15])

SEE ALSO

For a list of the AFP commands and their command code numeric values and constants, see Table 9-1 on page 9-9. To determine which AFP commands take the general `AFPCommand` format, see Table 9-2 on page 9-13. For a description of the AFP commands that you can send to a server and their required command block formats, see *Inside AppleTalk*, second edition, and the *AppleShare 3.0 Developer's Kit* version 3.0.

AFP Login Command Format

You use the login command format for the `AFPCommand` function to pass the `afpLogin` command to the .XPP driver to open a session with an AFP file server.

```
FUNCTION AFPCommand (thePBptr: XPPParmBlkPtr;
                     async: Boolean): OSErr;
```

`thePBptr` A pointer to the XPP parameter block format for the `afpLogin` command.

`async` A Boolean that specifies whether the function is to execute synchronously or asynchronously. Set the `async` parameter to `TRUE` to execute the function asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
←	`cmdResult`	`LongInt`	The AFP command result.
→	`csCode`	`Integer`	Always `afpCall` for this function.
←	`sessRefnum`	`Integer`	The session reference number.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`aspRetry`	`Byte`	The number of retries.
→	`cbSize`	`Integer`	The command buffer size.
→	`cbPtr`	`Integer`	A pointer to the command buffer.

↔	rbSize	Integer	On input, the reply buffer size. On return, the actual reply size.
→	rbPtr	Ptr	A pointer to the reply buffer.
→	afpAddrBlock	AddrBlock	The internet socket address of the server.
↔	afpSCBPtr	Ptr	A pointer to the SCB.
↔	afpAttnRoutine	Ptr	A pointer to an attention routine.

Field descriptions

`cmdResult` Four bytes of data returned from the server indicating the result of the AFP command.

`sessRefnum` The session reference number, which is a unique number that the .XPP driver assigns to the session and returns.

`aspTimeout` The interval in seconds that the .XPP driver waits between retries of the AFP command call.

`aspRetry` The number of times that the .XPP driver is to retry to execute the AFP command.

`cbSize` The size in bytes of the block of data that contains the command and its parameters to be sent to the server across the session. The size of the command block must not exceed the value of `aspMaxCmdSize` that the `ASPGetParms` function returns. For information on the `ASPGetParms` function, see the chapter "AppleTalk Session Protocol (ASP)."

`cbPtr` A pointer to the beginning of the command block that contains the AFP login command to be sent across the session to the server. The `cbSize` field value specifies the command block size. The first byte of the command block must contain the AFP login command. The following command block bytes contain the parameters for the command. For the definitions of the AFP commands and their command codes and parameters, see *Inside AppleTalk*, second edition, and the *AppleShare 3.0 Developer's Kit* version 3.0.

`rbSize` On input, the size in bytes of the reply buffer that is to hold the expected response to the AFP login command. On return, the actual size of the reply to the AFP command that the .XPP driver returned in the buffer.

`rbPtr` A pointer to the reply buffer.

`afpAddrBlock` The internet socket address of the server to which the command is to be sent.

`afpSCBPtr` A pointer to a session control block (SCB) that the .XPP driver requires to maintain an open session. The `scbMemSize` constant defines the size of the session control block. The memory that you allocate for the SCB must be nonrelocatable or locked because it belongs to the .XPP driver for the life of the session. Each session requires its own SCB.

`afpAttnRoutine` A pointer to a routine that the .XPP driver calls when it receives an attention request from the server. If you do not want the .XPP driver to call an attention routine, set this field to 0.

DESCRIPTION

To open a session with an AFP file server, you call the `AFPCommand` function and pass it the `afpLogin` command in the command block that you provide. You point to the command block from the XPP parameter block's `cbPtr` field. You specify the internet socket address of the server that you want to access as the value of the `afpAddrBlock` field.

In addition to allocating memory for the parameter block and the command block, you must provide a session control block (SCB) and pass the `AFPCommand` function a pointer to the SCB in the `afpSCBPtr` field. The .XPP driver uses the SCB internally to manage the session. Each session requires its own SCB. You must either allocate nonrelocatable memory for the session control block or lock the memory and not modify it for the duration of the session. The SCB size is defined by the constant `scbMemSize`. The memory belongs to the .XPP driver for the life of the session. You can reuse an SCB after either of the following events occurs:

- You have called an `AFPCommand` function using the general command format to specify an `afpLogout` command to close the session and the `AFPCommand` function has successfully completed execution.
- The server end of the session has closed the session or the .XPP driver has closed the session.

AFP includes an attention mechanism that allows the server to send an attention request to the workstation. For example, a file server can use this messaging system to notify all of the workstations that are using the file server that it is shutting down. The XPP parameter block for the login format includes a pointer to an attention routine.

When the .XPP driver receives an attention request, it sets the first 2 bytes of the SCB to the attention bytes from the packet. If you have provided an attention routine, the .XPP driver calls it. The .XPP driver also calls the attention routine when the session is closed by either the workstation or the server or AFP itself, for example, because the .XPP driver could not open a session before it exhausted the number of retries.

You code the attention routine in assembly language. Because the .XPP driver calls your attention routine at interrupt level, you must observe the following interrupt conventions in writing the attention routine:

- An attention routine can change registers A0 through A3 and D0 through D3.
- The routine must not call any Memory Manager routines.

The .XPP driver calls your attention routine with

- D0 (word) equal to the session reference number (`sessRefnum`) for that session. This is the number that AFP returns on completion of the `AFPCommand` function for the `afpLogin` command.
- D1 (word) equal to the attention bytes passed by the server or 0 if the session is closing.

To resume normal execution, your attention routine must return with an RTS (return from subroutine) instruction.

If you code your program in a high-level language, such as Pascal, you might not want to provide an attention routine written in assembly language. If you do not want to provide an attention routine, you can poll the attention bytes to determine if your application has received an attention request from the server. The attention bytes are the first 2 bytes of the session control block. When the server sends an attention request to the workstation, the .XPP driver receives the request and it sets the first 2 bytes of the SCB to the attention bytes from the packet. (When the session was opened, the .XPP driver set these bytes to 0.) If the first 2 bytes of the SCB are nonzero when your Pascal program polls them, the program will know that it has received an attention request from the server. Your program can handle the request and reset the SCB's attention bytes to 0. However, using this method to determine if the workstation has received an attention request from the server has limitations; two or more attention requests could be received between successive polls and only the last one would be preserved.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function. For more information on the `OpenDriver` function, see *Inside Macintosh: Devices.*

In the XPP parameter block for the `AFPCommand` function login format, the `afpSCBPointer` and `afpAttnRoutine` fields overlap with the beginning of the CCB and are modified by `AFPCommand` function.

The memory that you allocate for the XPP parameter block, command block, and reply buffer belongs to AFP until the function completes execution, after which you can reuse the memory or release it. However, the memory that you allocate for the SCB belongs to AFP for the life of the session. You must either allocate nonrelocatable memory for the SCB or lock the memory and not modify it for the duration of the session.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `AFPCommand` function from assembly language, call the `_Control` trap macro with a value of `afpCall` in the `csCode` field of the parameter block.

RESULT CODES

`aspBadVersNum`	–1066	The server cannot support the ASP version number
`aspBufTooSmall`	–1067	The command reply from the server is larger than the response buffer; ASP will fill the buffer and truncate the reply data
`aspNoMoreSess`	–1068	The .XPP driver cannot support another ASP session
`aspNoServers`	–1069	There is no server at the specified server address, or the server did not respond to the request
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspServerBusy`	–1071	The server cannot open another session
`aspSizeErr`	–1073	The size of the command block exceeds the maximum size of `aspMaxCmdSize`

SEE ALSO

For information on how to obtain the internet socket address of a server, see the chapter "Name-Binding Protocol (NBP)" in this book.

AFP Write Command Format

You use the write command format for the `AFPCommand` function to pass the `afpWrite` command to the .XPP driver to send a data block to the server.

```
FUNCTION AFPCommand (thePBptr: XPPParmBlkPtr;
                     async: Boolean): OSErr;
```

`thePBptr` A pointer to the XPP parameter block format for the `afpWrite` command.

`async` A Boolean that specifies whether the function is to execute synchronously or asynchronously. Set the `async` parameter to `TRUE` to execute the function asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
←	`cmdResult`	`LongInt`	The AFP command result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `afpCall` for this function.
→	`sessRefnum`	`Integer`	The session reference number.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`cbSize`	`Integer`	The command buffer size.
→	`cbPtr`	`Ptr`	The command buffer.
↔	`rbSize`	`Integer`	On input, the reply buffer size. On return, the actual reply size.
→	`rbPtr`	`Ptr`	A pointer to the reply buffer.
↔	`wdPtr`	`Ptr`	A pointer to the write data.

Field descriptions

`cmdResult` Four bytes of data returned from the server indicating the result of the AFP command.

`sessRefnum` The session reference number, which is a unique number that the .XPP driver assigns to the session and returns in response to an `afpLogin` command.

`aspTimeout` The interval in seconds that the .XPP driver waits between retries of the AFP command call.

`cbSize` The size in bytes of the block of data that contains the command and its parameters to be sent to the server across the session. The size of the command block must not exceed the value of `aspMaxCmdSize` that the `ASPGetParms` function returns. For information on the `ASPGetParms` function, see the chapter "AppleTalk Session Protocol (ASP)."

`cbPtr`	A pointer to the beginning of the command block buffer that contains the `afpWrite` command to be sent across the session to the server. The `cbSize` field value specifies the command block buffer size. The first byte of the command block must contain the AFP command. The following command block bytes contain the parameters for the command. The "Description" section that follows explains the command block structure that you use for the `afpWrite` command to be sent to the server.
`rbSize`	On input, the size in bytes of the reply buffer that is to hold the expected response to the AFP command. On return, the actual size of the reply that the .XPP driver returned in the buffer.
`rbPtr`	A pointer to the reply buffer.
`wdPtr`	A pointer to the write data buffer. The .XPP driver updates this field as it proceeds so that the field always points to the section of data that the .XPP driver is currently writing.

DESCRIPTION

After you open a session, you can use the `afpWrite` command to send a block of data to the server. The `AFPCommand` function format for the write command allows you to send more data than a single call to an `ASPUserWrite` function can send. Instead of using a write-data structure to specify the data to be sent, you specify the beginning of the data to be written and the size in bytes of the data as values within the command block. (You do not specify the size of the write data in the parameter block.)

The command block for the `afpWrite` command consists of the following fields. The byte offsets for these fields are relative to the location indicated by the command block pointer (`cbPtr`).

Command block

→	`cmdByte`	`Byte`	The AFP command code.
→	`startEndFlag`	`Byte`	A flag identifying offset relative to fork.
↔	`rwOffset`	`LongInt`	The offset within fork to begin writing.
↔	`reqCount`	`LongInt`	On input, requested size of data; on return, size of data actually written.

Field descriptions

`cmdByte`	The AFP command code, which is always `afpWrite` for this command.
`startEndFlag`	A 1-bit flag (the high bit of the byte) indicating whether the offset specified in the `rwOffset` field is relative to the beginning or the end of the fork: set the high bit to 0 to specify that the offset is relative to the beginning of the fork; set the high bit to 1 to specify that the offset is relative to the end of the fork. Set all other bits of this byte to 0.
`rwOffset`	The byte offset within the fork at which the write is to begin. The .XPP driver modifies the value of this field as it proceeds; the field always reflects the current value.

`reqCount`	On input, the size in bytes of the data to be written. On return, the actual size of the data that was written. The .XPP driver modifies the value of this field as it proceeds; the field always reflects the current value.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function. For more information on the `OpenDriver` function, see *Inside Macintosh: Devices.*

The memory that you allocate for the XPP parameter block, command block, and reply buffer belongs to AFP until the function completes execution, after which you can reuse the memory or release it.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `AFPCommand` function from assembly language, call the `_Control` trap macro with a value of `afpCall` in the `csCode` field of the parameter block.

RESULT CODES

`aspBufTooSmall`	–1067	The command reply from the server is larger than the response buffer (ASP will fill the buffer and truncate the reply data)
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspSessClosed`	–1072	The session reference number is valid, but the .XPP driver is in the process of closing the session
`aspSizeErr`	–1073	The size of the command block exceeds the maximum size of `aspMaxCmdSize`

SEE ALSO

See "AFP Command Block Record" on page 9-5 for the Pascal structure of the command block for an `afpWrite` command.

AFP Read Command Format

To read a block of data on an AFP file server, you use the read command format for the `AFPCommand` function, which passes the `afpRead` command to the .XPP driver.

```
FUNCTION AFPCommand (thePBptr: XPPParmBlkPtr;
                     async: Boolean): OSErr;
```

`thePBptr` A pointer to the XPP parameter block format for a particular group of AFP commands.

`async` A Boolean that specifies whether the function is to execute synchronously or asynchronously. Set the `async` parameter to `TRUE` to execute the function asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
←	`cmdResult`	`LongInt`	The AFP command result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `afpCall` for this function.
→	`sessRefnum`	`Integer`	The session reference number.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`cbSize`	`Integer`	The command buffer size.
→	`cbPtr`	`Ptr`	A pointer to the command buffer.
↔	`rbPtr`	`Ptr`	A pointer to the reply buffer.

Field descriptions

`cmdResult` Four bytes of data returned from the server indicating the result of the AFP command.

`sessRefnum` The session reference number, which is a unique number that the .XPP driver assigns to the session and returns in response to an `afpLogin` command.

`aspTimeout` The interval in seconds that the .XPP driver waits between retries of the AFP command call.

`cbSize` The size in bytes of the block of data that contains the command and its parameters to be sent to the server across the session. The size of the command block must not exceed the value of `aspMaxCmdSize` that the `ASPGetParms` function returns. The "Description" section that follows explains the command block structure that you use for the `afpRead` command. See the chapter "AppleTalk Session Protocol (ASP)" for information on the `ASPGetParms` function.

`cbPtr` A pointer to the beginning of the command block that contains the `afpRead` command. The `cbSize` field value specifies the command block size. The first byte of the command block must contain the AFP command. The following command block bytes contain the parameters for the command.

`rbPtr` A pointer to the reply buffer. The .XPP driver updates this field as it proceeds; the value of this field points to the section of the buffer into which the . XPP driver is currently reading data.

DESCRIPTION

After you open a session, you can use the `afpRead` command to read a block of data from the server. The `AFPCommand` function format for the read command allows you to read more data than you can through a single call to an `ASPUserCommand` function.

You use the command block buffer to pass the read command and its parameters to the .XPP driver. (You pass the size of the read data buffer in the command block, not in the parameter block.) The command block for the `afpRead` command consists of the following fields. The byte offsets for these fields are relative to the location indicated by the command block pointer (`cbPtr`).

Command block

→	`cmdByte`	`Byte`	The AFP command code.
↔	`rwOffset`	`LongInt`	The offset within fork to begin reading
↔	`reqCount`	`LongInt`	On input, size of the read data buffer; on return, size of data actually read into the buffer.
→	`newLineFlag`	`Byte`	A flag indicating whether the read is to be terminated at a specified character.
→	`newLineChar`	`Byte`	The character used to determine where the read should be terminated.

Field descriptions

`cmdByte` The AFP command code, which is always `afpRead` for this command.

`rwOffset` The byte offset within the fork at which the read is to begin. The .XPP driver modifies the value of this field as it proceeds; the field always reflects the current value.

`reqCount` On input, the requested size of the read data buffer. On return, the actual size of the data that was read. The .XPP driver modifies the value of this field as it proceeds; the field always reflects the current value.

`newLineFlag` A 1-bit flag (the high bit of the byte) indicating whether the read is to be terminated at a specified character: set the high bit to 0 to indicate that you are *not* specifying a new-line character in the `newLineChar` field; set the high bit to 1 to indicate that you *are* specifying a new-line character in the `newLineChar` field. Set all other bits to 0.

`newLineChar` A character from $00 to $FF inclusive that, when encountered in reading the fork, causes the read operation to terminate.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function. For more information on the `OpenDriver` function, see *Inside Macintosh: Devices.*

The memory that you allocate for the XPP parameter block, command block, and reply buffer belongs to AFP until the function completes execution, after which you can reuse the memory or release it.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `AFPCommand` function from assembly language, call the `_Control` trap macro with a value of `afpCall` in the `csCode` field of the parameter block.

RESULT CODES

`aspBufTooSmall`	–1067	The command reply from the server is larger than the response buffer (ASP will fill the buffer and truncate the reply data)
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspSessClosed`	–1072	The .XPP driver is in the process of closing the session
`aspSizeErr`	–1073	The size of the command block exceeds the maximum size of `aspMaxCmdSize`

SEE ALSO

See "AFP Command Block Record" on page 9-5 for the Pascal structure of the command block for an `afpRead` command.

Summary of AFP

Pascal Summary

Constants

```
CONST
   {.XPP Driver unit and reference numbers}
   xppUnitNum      =    40;         {XPP unit number}
   xppRefNum       =    -41;        {XPP reference number}
   afpCall         =    250;        {AFP call command. Command buffer }
                                    { contains code for the command to be }
                                    { executed}
   {AFP command codes}
   afpByteRangeLock  =  1;
   afpVolClose       =  2;
   afpDirClose       =  3;
   afpForkClose      =  4;
   afpCopyFile       =  5;
   afpDirCreate      =  6;
   afpFileCreate     =  7;
   afpDelete         =  8;
   afpEnumerate      =  9;
   afpFlush          =  10;
   afpForkFlush      =  11;
   afpGetDirParms    =  12;
   afpGetFileParms   =  13;
   afpGetForkParms   =  14;
   afpGetSInfo       =  15;
   afpGetSParms      =  16;
   afpGetVolParms    =  17;
   afpLogin          =  18;
   afpContLogin      =  19;
   afpLogout         =  20;
   afpMapID          =  21;
   afpMapName        =  22;
   afpMove           =  23;
   afpOpenVol        =  24;
```

```
   afpOpenDir        =  25;
   afpOpenFork       =  26;
   afpRead           =  27;
   afpRename         =  28;
   afpSetDirParms    =  29;
   afpSetFileParms   =  30;
   afpSetForkParms   =  31;
   afpSetVolParms    =  32;
   afpWrite          =  33;
   afpGetFlDrParms   =  34;
   afpSetFlDrParms   =  35;
   afpDTOpen         =  48;
   afpDTClose        =  49;
   afpGetIcon        =  51;
   afpGtIcnInfo      =  52;
   afpAddAPPL        =  53;
   afpRmvAPPL        =  54;
   afpGetAPPL        =  55;
   afpAddCmt         =  56;
   afpRmvCmt         =  57;
   afpGetCmt         =  58;
   afpAddIcon        =  192;     {special code for ASP write commands}

   {miscellaneous}
   xppLoadedBit      =  5;       {XPP bit in PortBUse}
   scbMemSize        =  192;     {size of memory for SCB}

   {constants for AFP command block startEndFlag & newLineFlag fields}
   xppFlagClr        =  0;
   xppFlagSet        =  128;
```

Data Types

Command Block for AFP Read and AFP Write Commands

```
TYPE AFPCommandBlock =
   PACKED RECORD
      cmdByte:          Byte;       {AFP command}
      startEndFlag:     Byte;       {flag identifying offset relative }
                                    { to fork}
      forkRefNum:       Integer;    {reserved}
      rwOffset:         LongInt;    {offset within fork to begin }
                                    { reading or writing}
```

```
      reqCount:         LongInt;    {on input, size of the data buffer; }
                                    { on return, size of the data actually }
                                    { written or read}
      newLineFlag:      Byte;       {for read, a flag indicating whether }
                                    { the read is to be terminated at }
                                    { a specific character; not used by }
                                    { write}
      newLineChar:      Char;       {character used to determine where }
                                    { the read is to be terminated; not }
                                    { used by write}
   END;
```

XPP Parameter Block for AFP

```
XPPPrmBlkType = (XPPPrmBlk...);
XPPSubPrmType = (ASPOpenPrm,ASPSubPrm);
XPPEndPrmType = (AFPLoginPrm,ASPEndPrm);

TYPE XPPParamBlock =
   PACKED RECORD
      qLink:               QElemPtr;      {reserved}
      qType:               Integer;       {reserved}
      ioTrap:              Integer;       {reserved}
      ioCmdAddr:           Ptr;           {reserved}
      ioCompletion:        ProcPtr;       {completion routine}
      ioResult:            OSErr;         {result code}
      cmdResult:           LongInt;       {command result (ATP user bytes)}
      ioVRefNum:           Integer;       {reserved}
      ioRefNum:            Integer;       {driver reference number}
      csCode:              Integer;       {command code}
   CASE XPPPrmBlkType OF
      XPPPrmBlk:
         (sessRefnum:      Integer;       {offset to session refnum}
         aspTimeout:       Byte;          {timeout for ATP}
         aspRetry:         Byte;          {retry count for ATP}
   CASE XPPSubPrmType OF
      ASPOpenPrm:
         (serverAddr:      AddrBlock;     {server address block}
         scbPointer:       Ptr;           {SCB pointer}
         attnRoutine:      Ptr);          {attention routine pointer}
      ASPSubPrm:
         (cbSize:          Integer;       {command block size}
         cbPtr:            Ptr;           {command block pointer}
         rbSize:           Integer;       {reply buffer size}
         rbPtr:            Ptr;           {reply buffer pointer}
```

```
   CASE XPPEndPrmType OF
      AFPLoginPrm:
         (afpAddrBlock:    AddrBlock;     {address block in AFP login}
         afPSCBPtr:        Ptr;           {SCB pointer in AFP login}
         afpAttnRoutine:   Ptr);          {attn routine pointer in AFP login}
      ASPEndPrm:
         (wdSize:          Integer;       {write data size}
         wdPtr:            Ptr;           {write data pointer}
         ccbStart:         ARRAY[0..295] OF Byte)));
                                          {command control block}
   END;

XPPParmBlkPtr = ^XPPParamBlock;

XPPPrmBlkType = (XPPPrmBlk...);
XPPSubPrmType = (ASPOpenPrm,ASPSubPrm);
XPPEndPrmType = (AFPLoginPrm,ASPEndPrm);
```

Routines

```
FUNCTION AFPCommand         (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
```

C Summary

Constants

```
enum {
   afpCall     = 250            /*AFP command (buffer has command code)*/
};

enum {                          /*AFPCall command codes*/
   afpFlush          = 10,
   afpForkFlush      = 11,
   afpGetDirParms    = 12,
   afpGetFileParms   = 13,
   afpGetForkParms   = 14,
   afpGetSInfo       = 15,
   afpGetSParms      = 16,
   afpGetVolParms    = 17,
   afpLogin          = 18,
   afpContLogin      = 19,
```

```
   afpLogout         = 20,
   afpMapID          = 21,
   afpMapName        = 22,
   afpMove           = 23,
   afpOpenVol        = 24,
   afpOpenDir        = 25,
   afpOpenFork       = 26,
   afpRead           = 27,
   afpRename         = 28,
   afpSetDirParms    = 29
};

enum {                          /*AFPCall command codes*/
   afpSetFileParms   = 30,
   afpSetForkParms   = 31,
   afpSetVolParms    = 32,
   afpWrite          = 33,
   afpGetFlDrParms   = 34,
   afpSetFlDrParms   = 35,
   afpDTOpen         = 48,
   afpDTClose        = 49,
   afpGetIcon        = 51,
   afpGtIcnInfo      = 52,
   afpAddAPPL        = 53,
   afpRmvAPPL        = 54,
   afpGetAPPL        = 55,
   afpAddCmt         = 56,
   afpRmvCmt         = 57,
   afpGetCmt         = 58,
   afpAddIcon        = 192       /*special code for ASP write commands*/
};

enum {
   xppLoadedBit      = 5,        /*XPP bit in PortBUse*/
   scbMemSize        = 192,      /*size of memory for SCB*/
   xppFlagClr        = 0         /*cs for AFPCommandBlock*/
};

enum {
xppFlagSet           = 128}      /*startEndFlag & NewLineFlag fields*/
};
```

Data Types

Command Block for AFP Read and AFP Write Commands

```
typedef struct {
   char  cmdByte;             /*AFP command*/
   char  startEndFlag;        /*flag identifying offset relative to fork*/
   short forkRefNum;          /*reserved*/
   long  rwOffset;            /*offset within fork to begin reading */
                              /* or writing*/
   long  reqCount;            /*on input, size of the data buffer; */
                              /* on return, size of the data actually */
                              /* written or read*/
   char  newLineFlag;         /*for read, a flag indicating whether the */
                              /* read is to be terminated at a specific */
                              /* character; not used by write*/
   char  newLineChar;         /*character used to determine where the read */
                              /* is to be terminated; not used by write*/
} AFPCommandBlock;
```

XPP Parameter Block for AFP

```
#define  XPPPBHeader\
   QElem    *qLink;           /*reserved*/\
   short    qType;            /*reserved*/\
   short    ioTrap;           /*reserved*/\
   Ptr      ioCmdAddr;        /*reserved*/\
   ProcPtr  ioCompletion;     /*completion routine*/\
   OSErr    ioResult;         /*result code*/\
   long     cmdResult;        /*command result*/\
   short    ioVRefNum;        /*reserved*/\
   short    ioRefNum;         /*.XPP driver reference number*/\
   short    csCode;           /*function code*/

typedef struct {
   XPPPBHeader
   short    sessRefnum;       /*offset to session refnum*/
   char     aspTimeout;       /*timeout for ATP*/
   char     aspRetry;         /*retry count for ATP*/
   short    cbSize;           /*command block size*/
   Ptr      cbPtr;            /*command block pointer*/
   short    rbSize;           /*reply buffer size*/
   Ptr      rbPtr;            /*reply buffer pointer*/
```

```
   short    wdSize;          /*write data size*/
   Ptr      wdPtr;           /*write data pointer*/
   char     ccbStart[296];   /*beginning of command control block */
                             /* (CCB)*/
}XPPPrmBlk;

typedef struct {
   XPPPBHeader
   short       sessRefnum;       /*offset to session refnum*/
   char        aspTimeout;       /*timeout for ATP*/
   char        aspRetry;         /*retry count for ATP*/
   short       cbSize;           /*command block size*/
   Ptr         cbPtr;            /*command block pointer*/
   short       rbSize;           /*reply buffer size*/
   Ptr         rbPtr;            /*reply buffer pointer*/
   AddrBlock   afpAddrBlock;     /*block in AFP login*/
   Ptr         afpSCBPtr;        /*SCB pointer in AFP login*/
   Ptr         afpAttnRoutine;   /*attn routine pointer in AFP login*/
   char        ccbFill[144];     /*beginning of command control block*/
}AFPLoginPrm;

typedef struct {
   XPPPBHeader
   short       sessRefnum;       /*offset to session refnum*/
   char        aspTimeout;       /*timeout for ATP*/
   char        aspRetry;         /*retry count for ATP*/
   AddrBlock   serverAddr;       /*server address block*/
   Ptr         scbPointer;       /*SCB pointer*/
   Ptr         attnRoutine;      /*attention routine pointer*/
} ASPOpenPrm;

typedef ASPOpenPrm *ASPOpenPrmPtr;
```

Routines

```
pascal OSErr AFPCommand     (XPPParmBlkPtr thePBptr, Boolean async);
```

Assembly-Language Summary

Constants

XPP Driver Unit Number

```
xppUnitNum          EQU         40                ;XPP unit number
xppLoadedBit        EQU         atpLoadedBit+1    ;XPP loaded bit number in
                                                  ; PortBUse
```

AFP Control Code

```
afpCall             EQU         250               ;AFP csCode
```

AFP Command Codes

```
afpByteRangeLock    EQU     1
afpVolClose         EQU     2
afpDirClose         EQU     3
afpForkClose        EQU     4
afpCopyFile         EQU     5
afpDirCreate        EQU     6
afpFileCreate       EQU     7
afpDelete           EQU     8
afpEnumerate        EQU     9
afpFlush            EQU     10
afpForkFlush        EQU     11
afpGetDirParms      EQU     12
afpGetFileParms     EQU     13
afpGetForkParms     EQU     14
afpGetSInfo         EQU     15
afpGetSParms        EQU     16
afpGetVolParms      EQU     17
afpLogin            EQU     18
afpContLogin        EQU     19
afpLogout           EQU     20
afpMapID            EQU     21
afpMapName          EQU     22
afpMove             EQU     23
afpOpenVol          EQU     24
afpOpenDir          EQU     25
afpOpenFork         EQU     26
```

```
afpRead             EQU     27
afpRename           EQU     28
afpSetDirParms      EQU     29
afpSetFileParms     EQU     30
afpSetForkParms     EQU     31
afpSetVolParms      EQU     32
afpWrite            EQU     33
afpGetFlDrParms     EQU     34
afpSetFlDrParms     EQU     35

afpDTOpen           EQU     48
afpDTClose          EQU     49
afpGetIcon          EQU     51
afpGtIcnInfo        EQU     52
afpAddAPPL          EQU     53
afpRmvAPPL          EQU     54
afpGetAPPL          EQU     55
afpAddCmt           EQU     56
afpRmvCmt           EQU     57
afpGetCmt           EQU     58

afpAddIcon          EQU   192     ;special code for ASP write commands
```

Miscellaneous

```
afpUseWrite    EQU   $C0          ;first call in range that maps to an
                                  ; ASPWrite
```

Data Structures

Parameter Block for General Command Format

18	`cmdResult`	long	AFP command result
26	`csCode`	word	command code; always `afpCall`
28	`sessRefnum`	word	session reference number
30	`aspTimeout`	byte	retry interval in seconds
32	`cbSize`	word	command buffer size
34	`cbPtr`	pointer	command buffer
38	`rbSize`	word	reply buffer size and actual reply size
40	`rbPtr`	pointer	reply buffer pointer
44	`wdSize`	word	write data size
46	`wdPtr`	pointer	write data pointer
50	`ccbStart`	record	beginning of memory for CCB

Parameter Block for Login Command Format

18	`cmdResult`	long	AFP command result
26	`csCode`	word	command code; always `afpCall`
28	`sessRefnum`	word	session reference number
30	`aspTimeout`	byte	retry interval in seconds
31	`aspRetry`	byte	number of retries
32	`cbSize`	word	command buffer size
34	`cbPtr`	pointer	command buffer
38	`rbSize`	word	reply buffer size and actual reply size
40	`rbPtr`	pointer	reply buffer pointer
44	`afpAddrBlock`	long	server address block
48	`afpSCBPtr`	pointer	SCB pointer
52	`afpAttnRoutine`	pointer	attention routine pointer
50	`ccbStart`	record	beginning of memory for CCB

Parameter Block for AFP Write Command Format

18	`cmdResult`	long	AFP command result
26	`csCode`	word	command code; always `afpCall`
28	`sessRefnum`	word	session reference number
30	`aspTimeout`	byte	retry interval in seconds
32	`cbSize`	word	command buffer size
34	`cbPtr`	pointer	command buffer
38	`rbSize`	word	reply buffer size and actual reply size
40	`rbPtr`	pointer	reply buffer pointer
44	`wdSize`	word	used internally
46	`wdPtr`	pointer	write data pointer, updated
50	`ccbStart`	record	beginning of memory for CCB

Command Block for the AFP Write Command

0	`cmdByte`	byte	AFP command code
1	`startEndFlag`	byte	start/end flag
4	`rwOffset`	long	offset within fork to begin writing
8	`reqCount`	long	on input, requested size of data; on return, size of data actually written
12	`newLineFlag`	byte	new line flag
13	`newLineChar`	byte	new line character

Parameter Block for AFP Read Command Format

18	`cmdResult`	long	AFP command result
26	`csCode`	word	command code; always `afpCall`
28	`sessRefnum`	word	session reference number
30	`aspTimeout`	byte	retry interval in seconds
32	`cbSize`	word	command buffer size
34	`cbPtr`	pointer	command buffer
38	`rbSize`	word	used internally
40	`rbPtr`	pointer	reply buffer pointer (updated)
50	`ccbStart`	record	beginning of memory for CCB

Command Block for the AFP Read Command

0	`cmdByte`	byte	AFP command code
1	`startEndFlag`	byte	flag identifying offset relative to fork
4	`rwOffset`	long	offset within fork to begin reading
8	`reqCount`	long	on input, requested size of data; on return, size of data actually read into the buffer

Result Codes

`aspBadVersNum`	–1066	The server cannot support the ASP version number
`aspBufTooSmall`	–1067	The command reply from the server is larger than the response buffer (ASP will fill the buffer and truncate the reply data)
`aspNoMoreSess`	–1068	The .XPP driver cannot support another ASP session
`aspNoServers`	–1069	There is not a server at the specified server address, or the server did not respond to the request
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspServerBusy`	–1071	The server cannot open another session
`aspSessClosed`	–1072	The .XPP driver is in the process of closing the session
`aspSizeErr`	–1073	The size of the command block exceeds the maximum size of `aspMaxCmdSize`
`afpParmError`	–5019	The AFP command block size is equal to 0 (this error is also returned when the command block is equal to 0 or $FF [255] or `GetSrvrStatus` [15])