APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION

290 SW 43rd Street
Renton, WA 98055
206-251-6548

AppleTalk Manager Update

APDA# KMSAMU

APDA Macintosh AppleTalk Update
Release Notes
February 25, 1987

The APDA Macintosh AppleTalk Update consists of one 400K diskette, a document entitled "AppleTalk Manager Update, APDA Draft," and these release notes. The disk contains updates and additions to the current Macintosh AppleTalk utilities. The document provides technical details of many of the changes. It is a "pre-release" of a chapter to be published in *Inside Macintosh* volume 5; future updates to it will appear there. These release notes are meant to summarize and supplement that document.

Disk Contents: The APDA Macintosh AppleTalk Update Disk includes the following:

> (1) Version 4.0 of the AppleTalk High-Level ("Pascal") Interface for MPW. This version is a maintenence release which fixes some known bugs.
>
> (2) Version 48 of the AppleTalk drivers. This version, which is the one used in the AppleShare server, provides many additional features over the version provided in the MacPlus ROM (version 40).
>
> (3) Version 1.1 of the new Extended Protocol Package (XPP) driver. This driver, used in the AppleShare workstation, implements the workstation side of the AppleTalk Session Protocol (ASP).
>
> (4) MPW Pascal and C interfaces to the XPP driver.

Each of these components is summarized in the following pages. Since the distribution disk is an MFS volume, you may wish to copy it to an HFS volume before installation.

Important: As stated in your APDA membership agreement, these utilities are provided for development purposes only; no license to distribute in any way is implied. Please contact Apple Software Licensing if you wish to distribute any of the enclosed material.

AppleTalk High-Level Interface Version 4.0

Version 4.0 of the AppleTalk "Pascal" Interface is a maintenance release. It replaces the current version, 3.4. There are three major changes to this version:

(1) elimination of the use of location $E0
(2) elimination of the 'atpl' ("{Libraries}AppleTalk") resource
(3) bug fixes.

Each of these changes is described below. Note that there is no added support for any of the new calls available in the new AppleTalk drivers (version 48). See the section on "future plans."

Use of location $E0: Previous versions of the Pascal Interface used low memory location $E0. This created a number of problems. The use of this location has been eliminated.

The 'atpl' resource: Previous versions of the Pascal Interface included a resource of type 'atpl' which had to be included in either the system file or the app's resource fork. This resource was shipped as the file {Libraries}AppleTalk under MPW 1.0. It always took up about 5K on disk and in memory, regardless of how much of it was needed. Questions arose as to how to release the resource when it was no longer needed. This resource was called by a small bit of glue linked into your application out of Interface.o.

The current release eliminates the use of this resource altogether. The glue in interface.o has been replaced with the actual code from the resource. Thus the actual code is now linked directly into your application. Since the linker only includes that code which is needed, in general only 1-2K will be added to the size of your application's code. Since 5K will be deleted by the elimination of the 'atpl' resource, there should be a net space savings, both on disk and in memory. Issues such as how to free the resource are also eliminated.

Bug fixes: A number of small bugs have been fixed. These include:

(1) DDPRdCancel could crash if you tried to cancel a DDPRead that wasn't there

(2) ATPReqCancel and ATPRspCancel now ignore the async flag and always execute synchronously. This is because the ABRecord passed in these calls is the ABRecord of the call to be cancelled, not a new ABRecord for the cancel call. Thus there is no way to indicate completion of the call itself if it were asynchronous.

Installation procedure: Included on the release disk, in folder "PI4.0" is an MPW script which replaces the current glue in {Libraries}Interface.o with the new routines. It is called "NewInterface". To execute it, just set the current directory to that folder (pi4.0) and type NewInterface. This will copy the routines from file "ABPackage.o" into {Libraries}Interface.o (note that executing NewInterface twice will result in a number of warnings). Alternatively, you could just include the file ABPackage.o on your link command line before Interface.o. This will cause a number of duplicate label warnings, however.

Following installation, all programs using the Pascal Interface should be re-linked, and the 'atpl' resource can be removed from all disks.

Future plans: We intend this to be the final release of the current-style interface. We will continue to support this for some time. However, in the long run, we intend to move to a more straightforward and simple interface design. This interface, which will also support all of the new AppleTalk functionality, will be a parameter-block style interface, much like the current low-level file system interface detailed in *Inside Mac,* volume 4. Additional functionality will be added only where necessary to support operations that can not be performed in a higher-level language (for instance socket listeners).

Developers will be free to continue to use the old style interface for MacPlus-style AppleTalk functionality. However we believe in the long run it will be advantageous to move to the new style interface.

We expect this new style interface to be initially available in 2-3 months. Most calls will be essentially the same format as the high-level XPP calls included with this release.

AppleTalk Drivers Version 48

Version 48 of the AppleTalk drivers contains significant enhancements over the current version (40 in the MacPlus ROMs, and 41 in system file 3.2). These enhancements include new calls and additional resources (e.g. all dynamic sockets can be opened simultaneously; more than one NBP request can be outstanding at any one time). These drivers are used by the AppleShare server. We envision these drivers to be used in place of the MacPlus ROM drivers in two circumstances: (1) where one or more of the new calls is required and (2) in servers where additional resources are needed. These new drivers are completely compatible with the MacPlus ROM drivers, and can be used in any situation where the ROM drivers were used previously.

The developer should be aware however that use of these drivers on a MacPlus or 512Ke will result in the loss of about 11K of system heap, since they completely replace the ROM drivers and since more memory is required to provide the additional resources. This may require the system heap to be grown, depending on the application.

On an original ROM 512K Mac, these new drivers can be placed in the system file and will load on the first _Open call. On a MacPlus or 512Ke, however, ROM drivers supersede those in the system file, so the new drivers must be loaded in some other method. This release provides a system file INIT resource, INIT 22, which, for MacPlus ROMs, opens the new drivers out of a file called "AppleTalk" in the system folder. If the file is not there, the INIT resource does nothing. Note that this INIT must be installed in the system file so that it runs before any application INIT's (INIT 31's) which might open AppleTalk and start up a request. Apple intends to ship this INIT as part of future system files; however, for now, the developer must install it himself.

These new drivers are distributed as five resources in the file "AppleTalk" within the folder "newMPPATP". These are DRVR 9 and 10, and NBPC 1 and 2. In addition, this file contains a resource of type 'mppc'. This is a new resource used to pass information to the INIT. Its format will be specified in the future. This resource, as provided, tells the INIT to open AppleTalk in server mode (i.e. pick a server node number, a process which takes about 10 seconds). If you wish to have AppleTalk opened in workstation mode, do not include this resource in the "AppleTalk" file.

The INIT 22 resource is also included in the "AppleTalk" file. To use the new AppleTalk drivers for development purposes, you can install this INIT in your system file and copy the file "AppleTalk" into your system folder, deleting the 'mppc' resource if desired.

For development purposes, installation of the INIT resource can be done using ResEdit or the equivalent. However, for shipping products, installation from the product disk onto the user's disk should be done using the Apple Installer. The current version of that installer is provided in the "newMPPATP" folder, along with the installation script "AppleTalk INIT Script". This script, which should be run off the product disk containing the "AppleTalk" file in the system folder, installs the INIT into the user's system file, copies the "AppleTalk" file into the system folder, and sets the boot blocks to a minimum of 58K (this is also the size used on the AppleShare workstation disk).

Assembly language equates for the new calls in these drivers are distributed in the file "nATalkEqu.a" in the same folder. This file also contains assembly equates for the new XPP driver.

XPP Version 1.1

The XPP (Extended Protocol Package) driver implements the workstation side of the AppleTalk Session Protocol (ASP), and a small portion of the AppleTalk Filing Protocol (AFP). It is installed in the system file and used in the same manner as the other AppleTalk drivers. XPP is automatically installed by the AppleShare workstation installer. Developers wishing to use XPP without AppleShare workstation software will have to install it in the system file themselves.

The XPP driver is resource DRVR 40, and is contained in the file XPP1.1 in the folder "XPP". The assembly language interface to this driver is included in the file "nAtalkEqu.a" in the folder "newMPPATP".

XPP High-Level Interface

A high-level, parameter block style interface to XPP is provided for both MPW Pascal and MPW C. It is distributed in three files in the "XPP" folder. The files "nAppleTalk.p" and "ncAppleTalk.h" are the Pascal and C interfaces, respectively, to the XPP driver. The file nAppleTalk.a.o is the object file containing the actual interface code. This file should be included on the link command line. Also included in this folder are two example MPW tools: "exOpenSession.p" and "exOpenSession.c". These tools simply issue an OpenSession call to XPP, as specified by their arguments.

AppleTalk Manager Update

APDA Draft

Apple Technical Publications Department

January 23, 1987

Table of Contents

APPLETALK MANAGER UPDATE

The AppleTalk Manager has been enhanced through the implementation of new protocols and increased functionality of the existing interface. This chapter describes these enhancements beginning with a brief summary of the changes that have been made. The remainder of the chapter provides detailed information about these changes.

Summary of Changes

The AppleTalk Manager provides services that allow Macintosh programs to interact with clients in devices connected to an AppleTalk network. The following is a brief summary of the changes that have been made to the AppleTalk Manager interface.

- At open time, the .MPP driver can be told to pick a node number in the server range. This is a more time consuming but more thorough operation than selecting a node number in the workstation range, and is required for devices acting as servers.
- Multiple concurrent NBP requests are now supported (just as multiple concurrent ATP requests have been supported). The KillNBP command has been implemented to abort an outstanding NBP request.
- ATP requests can now be sent *through* client-specified sockets, instead of having ATP pick the socket itself.
- The ability to send packets to one's own node is supported (although this functionality is, in the default case, disabled).
- Two new ATP abort calls have been added: KillSendReq and KillGetReq. KillSendReq is functionally equivalent to RelTCB, although its arguments are different. KillGetReq is a new call for aborting outstanding GetRequests.
- Additional machine-dependent resources have been added to support , for example, more dynamic sockets and more concurrent ATP requests.
- A new protocol called the Echo Protocol (EP) is supported.
- A new driver, .XPP has been added. The .XPP driver implements the workstation side of the AppleTalk Session Protocol (ASP), and a small portion of the AppleTalk Filing Protocol.

To determine if you are running on a machine that supports these enhanced features, check the version number of the .MPP driver (at offset DCtlQueue+1 in the Device Control Entry). A version number of 48 (NCVersion) or greater indicates the presence of the new drivers.

> *Note:* With the exception of the XPP calls, there is currently no high-level support for any of the new calls. They can however, be accessed through appropriate PBControl calls.

CHANGES TO THE APPLETALK MANAGER

Changes to the AppleTalk manager increase functionality and resourses. Picking a node address in the server range, sending packets to one's own node, multiple concurrent NBP requests, sending ATP requests through a specified socket and two new ATP calls are discussed in this section.

Picking a node address in the server range

Normally upon opening, the node number picked by the AppleTalk manager will be in the node number range ($01-$7F). It is possible to indicate that a node number in the server range ($80-$FE) is desired. Picking a number in the server range is a more time consuming but more thorough process and is required for server nodes, since it greatly decreases the possibility of a node number conflict.

To open AppleTalk with a server node number, an extended open call is used. An extended open call is indicated by having the immediate bit set in the Open trap itself. In the extended open call, bit 7 of the extension longword field (ioMix) indicates whether a server or workstation node number should be picked. Set this bit to 1 to request a server node number. The rest of this field should be zero, as should all other unused fields in the queue element. A server node number can only be requested on the first Open call to the .MPP driver.

Sending packets to one's own node

Upon opening, the ability to send a packet to one's own node (intra-node delivery) is disabled. This feature of the AppleTalk Manager can be manipulated through the SetSelfSend function. Once enabled, it is possible, at all levels, to send packets to entities within one's own node. An example of where this might be desirable is an application sending data to a print spooler which is actually running in the background on the same node.

Note that enabling (or disabling) this feature affects the entire node and should be performed with care. For instance, a desk accessory may not expect to receive names from within its own node as a response to an NBP lookup; enabling this feature from an application could break the desk accessory. All future programsshould be written with this feature in mind.

SetSelfSend function

Parameter Block

```
—>   26   csCode word        ; always SetSelfSend
—>   28   NewSelfFlag byte   ; new SelfSend flag
<—   29   OldSelfFlag byte   ; old SelfSend flag
```

SetSelfSend enables or disables the intra-node delivery feature of the AppleTalk Manager. If NewSelfFlag is non-zero, the feature will be enabled; otherwise it will be disabled. The previous value of the flag will be returned in OldSelfFlag.

Result Codes	noErr	No error

AppleTalk Transaction Protocol Changes

Sending an ATP request through a specified socket

ATP requests can now be sent through client-specified sockets. ATP previously would open a dynamic socket, send the request through it, and close the socket when the request completed. The client can now choose to send a request through an already-opened socket. This also allows more than one request to be sent per socket. A new call, NSendRequest, has been added for this purpose. The function of the old SendRequest call itself remains unchanged.

NSendRequest function

Parameter block

--->	18	userData	longword	;user bytes
<---	22	reqTID	word	;transaction ID used in requet
--->	26	csCode	word	;always sendRequest
--->	28	atpSocket	byte	;socket to send request on
<-->	29	atpFlags	byte	;control information
--->	30	addrBlock	longword	;destination socket address
--->	34	reqLength	word	;request size in bytes
--->	36	reqPointer	pointer	;pointer to request data
--->	40	bdsPointer	pointer	;pointer to response BDS
--->	44	numOfBuffs	byte	;number of responses expected
--->	45	timeOutVal	byte	;timeout interval
<---	46	numOf Resps	byte	;number of responses received
<-->	47	retryCount	byte	;number of retries
<---	48	intBuff	word	;used internally

The NSendRequest call is functionally equivalent to the SendRequest call, however NSendRequest allows you to specify, in the atpSocket field, the socket *through* which the request is to be sent. This socket must have been previously opened through an OpenATPSkt request (otherwise a badATPSkt error will be returned). Note that NSendRequest requires two additional bytes of memory at the end of the parameter block, immediately following the retryCount. These bytes are for the internal use of the AppleTalk Manager and should not be modified while the NSendRequest call is active.

There is a machine-dependent limit as to the number of concurrent NSendRequests that can be active on a given socket. If this limit is exceeded, the error tooManyReqs will be returned.

Note that one additional difference between SendRequest and NSendRequest is that an NSendRequest can only be aborted by a KillSendReq call (see below), whereas a SendRequest can be aborted by either a RelTCB or KillSendReq call.

Result Codes	noErr	No error
	reqFailed	Retry count exceeded
	tooMany Reqs	Too many concurrent requests
	noData Area	Too many outstanding ATP calls
	reqAborted	Request cancelled by user

Aborting ATP SendRequests

The RelTCB call is still supported, but only for aborting SendRequests. To abort NSendRequests, a new call, KillSendReq has been added. This call will abort both SendRequests and NSendRequests. KillSendReq's only argument is the queue element pointer of the request to be aborted. The queue element pointer is passed at the offset of the KillSendReq queue element specified by AKillQE1.

KillSendReq function

Parameter block

```
    -->    26    csCode      word       ; always KillSendReq
    -->    44    AKillQEl    pointer    ; pointer to queue element
```

KillSendReq is functionally equivalent to RelTCB, except that it takes different arguments and will abort both SendRequests and NSendRequests. To abort one of these calls, place a pointer to the queue element of the call to abort in AKillQEl and issue the KillSendReq call.

Result Codes	noErr	No error
	cbNotFound	AKillQEl does not point to a SendReq or NSendReq queue element

Aborting Get Requests

ATP GetRequests can now be aborted through the KillGetReq call. This call looks and works just like the KillSendReq call, and is used to abort a specific GetRequest call. Previously it was necessary to close the socket to abort all GetRequest calls on the socket.

KillGetReq function

Parameter block

```
    -->    26    csCode      word       ; always KillGetReq
    -->    44    AKillQEl    pointer    ; pointer to queue element
```

KillGetReq will abort a specific outstanding GetRequest call (as opposed to closing the socket which aborts all outstanding GetRequests on that socket). The call will be completed with a reqAborted error. To abort a GetRequest, place a pointer to the queue element of the call to abort in AKillQEl and issue the KillGetReq call.

Result Codes	noErr	No error
	cbNotFound	AKillQEl does not point to a GetReq queue element

Name Binding Protocol Changes

Multiple Concurrent NBP Requests

NBP now supports multiple concurrent active requests. Specifically, a number of LookupNames, RegisterNames and ConfirmNames can all be active concurrently. The maximum number of concurrent requests is machine dependent; if it is exceeded the error tooManyReqs will be returned. Active requests can be aborted by the KillNBP call.

KillNBP function

Parameter block

->	26	csCode	word	; always KillNBP
->	28	NKillQEl	pointer	; pointer to queue element

KillNBP is used to abort an outstanding LookupName, RegisterName or ConfirmName request. To abort one of these calls, place a pointer to the queue element of the call to abort in NKillQEl and issue the KillNBP call. The call will be completed with a ReqAborted error.

Result Codes	noErr	No error
	cbNotFound	NKillQEl does not point to a valid NBP queue element

Variable resources

All dynamic sockets ($80 through $FE) can now be opened concurrently in addition to twelve (12) static sockets.

The following resources have also been increased:

Number of protocols handlers (MPP)
Number of concurrent SendRequests (ATP)
Number of ATP sockets
Number of concurrent XO Send Responses (ATP)
Number of data areas (ATP)
Number of concurrent NBP requests

NEW APPLETALK PROTOCOLS

The following protocols have been added to the AppleTalk Manager:

- Echo Protocol
- AppleTalk Session Protocol (workstation side)
- AppleTalk Filing Protocol (small portion of the workstation side)

The AppleTalk system architecture consists of a number of protocols arranged in layers. Each protocol in a specific layer provides services to higher-level layers (knowns as the protocol's clients) by building on the services provided by the lower-level layers. Figure 1 shows the AppleTalk Protocols and their corresponding network layers.

In Figure 1, the lines indicate the interaction between the protocols. Note that like the Routing Table Maintenance Protocol, the Echo Protocol is not directly accessible to Macintosh programs.

The details of these protocols are provided in *Inside AppleTalk*.

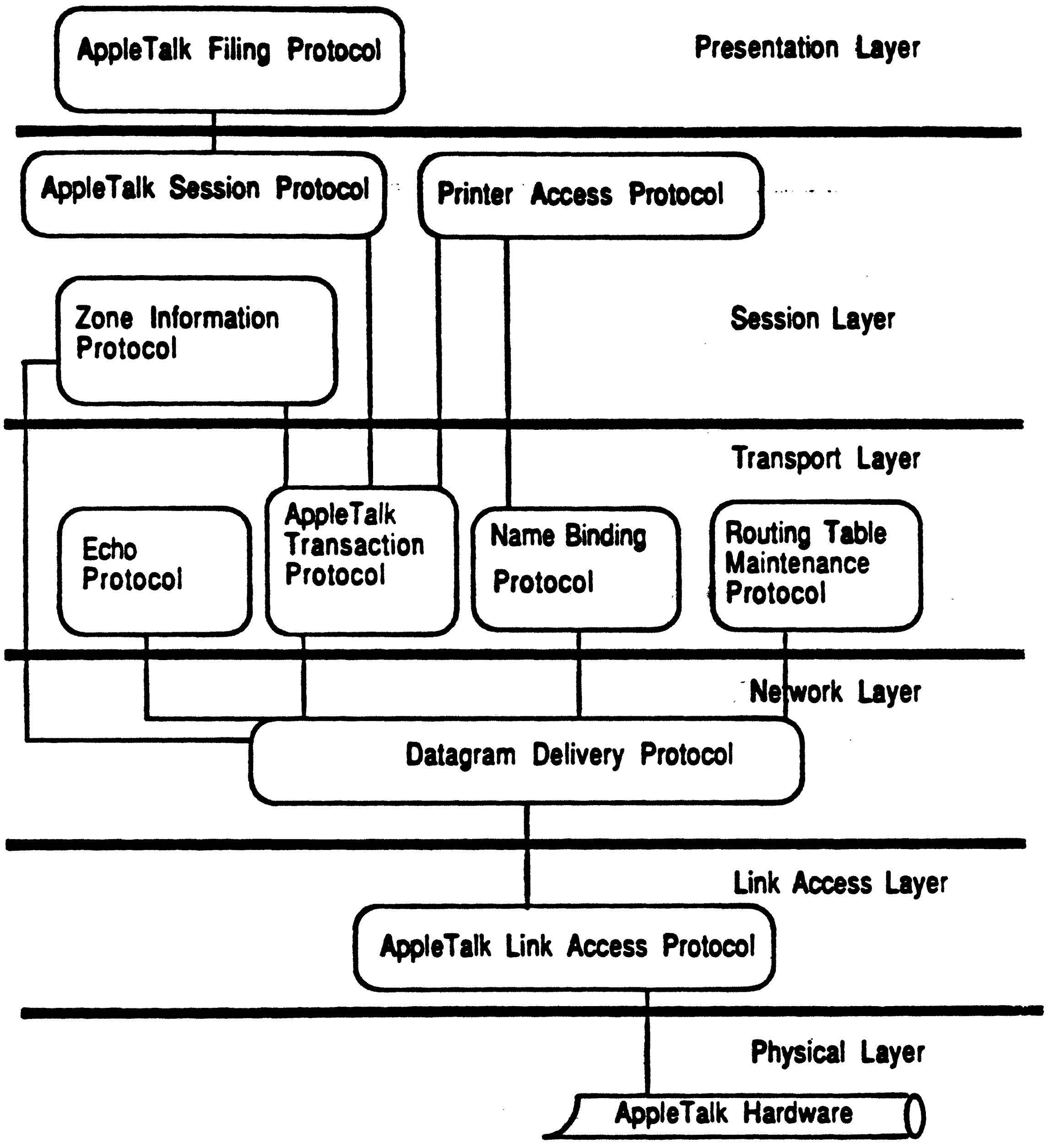
AppleTalk Filing Protocol
Presentation Layer
AppleTalk Session Protocol
Printer Access Protocol
Zone Information Protocol
Session Layer
Transport Layer
Echo Protocol
AppleTalk Transaction Protocol
Name Binding Protocol
Routing Table Maintenance Protocol
Network Layer
Datagram Delivery Protocol
Link Access Layer
AppleTalk Link Access Protocol
Physical Layer
AppleTalk Hardware

Figure 1. AppleTalk Protocols and OSI Nework Layers

Echo Protocol

The **Echo Protocol** (EP) provides an echoing service through static socket number 4 known as the **echoer socket**. The **echoer** listens for packets received through this socket. Any correctly formed packet sent to the echoer socket on a node, will be echoed back to its sender.

This simple protocol can be used for two important purposes:

1. EP can be used by any Datagram Delivery Protocol (DDP) client to determine if a particular node, (known to have an echoer) is accessible over an internet.
2. EP is useful in determining the average time it takes for a packet to travel to a remote node and back. This is very useful in developing client-dependent heuristics for estimating the timeouts to be specified by clients of ATP, ASP and other protocols.

Programs can not access EP directly via the AppleTalk Manager. The EP implementation exists solely to respond to EP requests sent by other nodes. EP is a DDP client residing on statically-assigned socket 4, the echoing socket. Clients wishing to send EP requests (and receive EP responses) should use the Datagram Delivery Protocol (DDP) to send the appropriate packet.

AppleTalk Session Protocol

The **AppleTalk Session Protocol** (ASP) provides for the setting up, maintaining and closing down of a **session**. A session is a logical relationship between two network entities, a **workstation** and a **server**. The workstation tells the server what to do and the server responds with the appropriate actions. ASP makes sure that the sessions dialog is maintained in the correct sequence and that both ends of the conversation are properly participating.

ASP will generally be used between two communicating network entities where one is providing a service to the other (i.e. a server is providing a service to a workstation) and the service provided is *state-dependent*. That is, the response to a particular request from an entity is dependent upon other previous requests from that entity. For example, a request to read bytes from a file is dependent upon a previous request to open that file in the first place. However, a request to return the time of day is independent of all such previous requests.

When the service provided is state-dependent, requests must be delivered to the server in the same order as generated by the workstation. ASP guarantees requests are delivered to the server in the order in which they are issued, and that duplicate requests are never delivered (another requirement of state-dependent service).

What ASP does

ASP is an asymetric protocol, providing one set of services to the workstation and a different set of services to the server.

ASP workstation clients initiate (open) sessions, send requests (commands) on that session and close sessions down. ASP server clients receive and respond (through command replies) to these requests. ASP guarantees that these requests are delivered in the same order as they are made, and without duplication. ASP is also responsible for closing down the session if one end dies or becomes unreachable and will inform its client (either server or workstation) of the action.

ASP also provides various additional services, such as allowing a workstation to obtain server status information without opening a session to a server, writing blocks of data from the workstation to the server end of the session, and the ability for a server to send an attention message to the workstation.

ASP assumes that the workstation client has a mechanism for looking up the network address of the server with which it wants to set up a session (generally this is done using the AppleTalk Name Binding Protocol).

Both ends of the session periodically check to see that the other end of the session is still responsive. If one end dies or becomes unreachable the other end closes the session.

ASP is a client of ATP and calls ATP for transport services.

What ASP does not do

ASP does not:

- allow the server to send commands to the workstation. The server is allowed to alert the workstation through the server's *attention* mechanism only.
- understand or interpret the syntax or the semantics of the commands sent to the server by the workstation.
- provide a user authentication (password) mechanism.
- insure that consecutive commands *complete* in the order in which they were sent (and delivered) to the server.

Note: The XPP driver does implement the workstation side of the AppleTalk Filing Protocol login command.

AppleTalk Filing Protocol

The AppleTalk Filing Protocol (AFP) allows a workstation on an AppleTalk network to access files on an AFP file server. AFP specifies a complex remote filing system containing user authentication and an access control mechanism that supports volume and folder-level access rights. For details of AFP, refer to the *AFP Draft Proposal*.

EXTENDED PROTOCOL PACKAGE DRIVER

The Extended Protocol Package (XPP) driver is intended to implement several AppleTalk communication protocols in the same package for ease of use. The .XPP driver currently consists of two modules that operate on two levels: the low-level implements the workstation side of AppleTalk Session Protocol, and the high-level implements (a small portion of) the workstation side of the AppleTalk Filing Protocol.

This driver adds functionality to the AppleTalk manager by providing services additional to those provided in the .MPP and .ATP drivers. Figure 2 shows the Macintosh AppleTalk drivers and the protocols accessible through each driver.

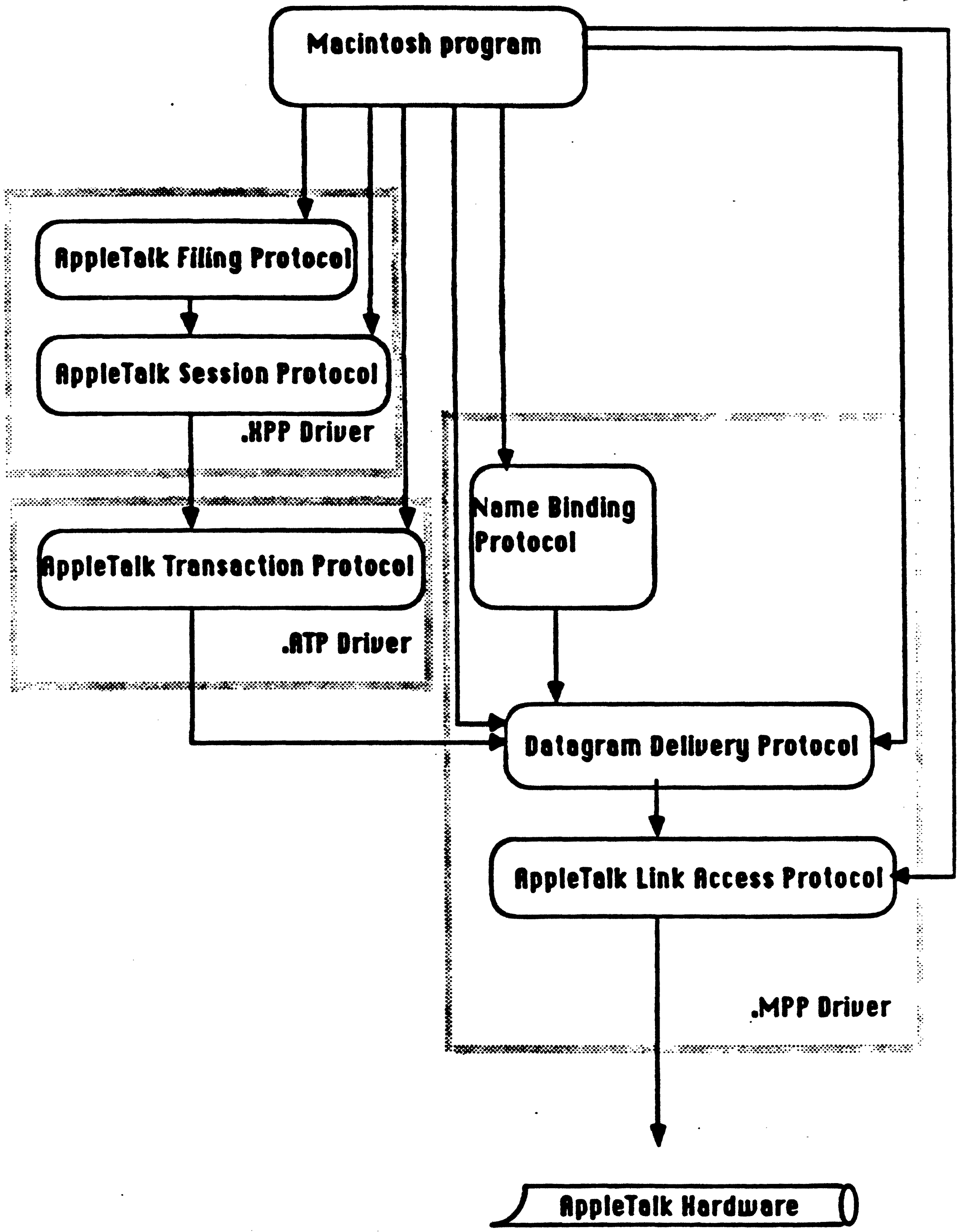
Macintosh program
AppleTalk Filing Protocol
AppleTalk Session Protocol
.XPP Driver
AppleTalk Transaction Protocol
.ATP Driver
Name Binding Protocol
Datagram Delivery Protocol
AppleTalk Link Access Protocol
.MPP Driver
AppleTalk Hardware

Figure 2. Macintosh AppleTalk Drivers

The .XPP driver maps an AFP call from the client workstation into one or more ASP calls. XPP provides one client-level call for AFP.

The implementation of AFP in the .XPP driver is very limited. Most calls are a very simple one-to-one mapping from an AFP call to an ASP command without any interpretation of the syntax of the AFP command by the .XPP driver. Refer to the *Mapping AFP commands* section of this document for further information.

Version

The .XPP driver supports ASP Version (hex) $100, as described in *Inside AppleTalk*.

Error reporting

Errors are returned by the .XPP driver in the ioResult field of the Device Manager Control calls.

The error conditions reported by the .XPP driver may represent the unsuccessful completion of a routine in more than just one process involved in the interaction of the session. System-level, .XPP driver, AppleTalk, and server errors can all turn up in the ioResult field. Note that an ASP server error actually results from the activity of the server end of the transaction but is reported through the .XPP driver.

AFP calls also return codes indicating the unsuccessful completion of AFP commands in the Command Result field of the parameter block (described below).

An application using the .XPP driver should respond appropriately to error conditions reported from the different parts of the interaction. As shown in figure 3, the following errors can be returned in the ioResult field:

1. System-level errors

   System errors returned by the .XPP driver indicate such conditions as the driver not being open or a specific system call not being supported. For a complete list of result codes returned by the Macintosh system software, refer to *Inside Macintosh,* Appendix A.

2. XPP errors (for example, *session not opened*)

   The .XPP driver can also return errors resulting from its own activity (for example, the referenced session isn't open). The possible .XPP driver errors returned are listed in the .XPP driver results codes section with each function that can return the code.

3. AppleTalk Errors (returned from lower-level protocols)

   XPP may also return errors from lower-level protocol, (for example, socket not open).

   Possible error conditions and codes are described in *Inside Macintosh,* Volume 2, Chapter 10, "The AppleTalk Manager".

4. An ASP-specific error could be returned from an ASP server in response to a failed *OpenSession* call. Errors of this type, returned by the server to the workstation, are

documented both in *Inside AppleTalk* Section 11, ""AppleTalk Session Protocol"and in the .XPP driver results code section of this document.

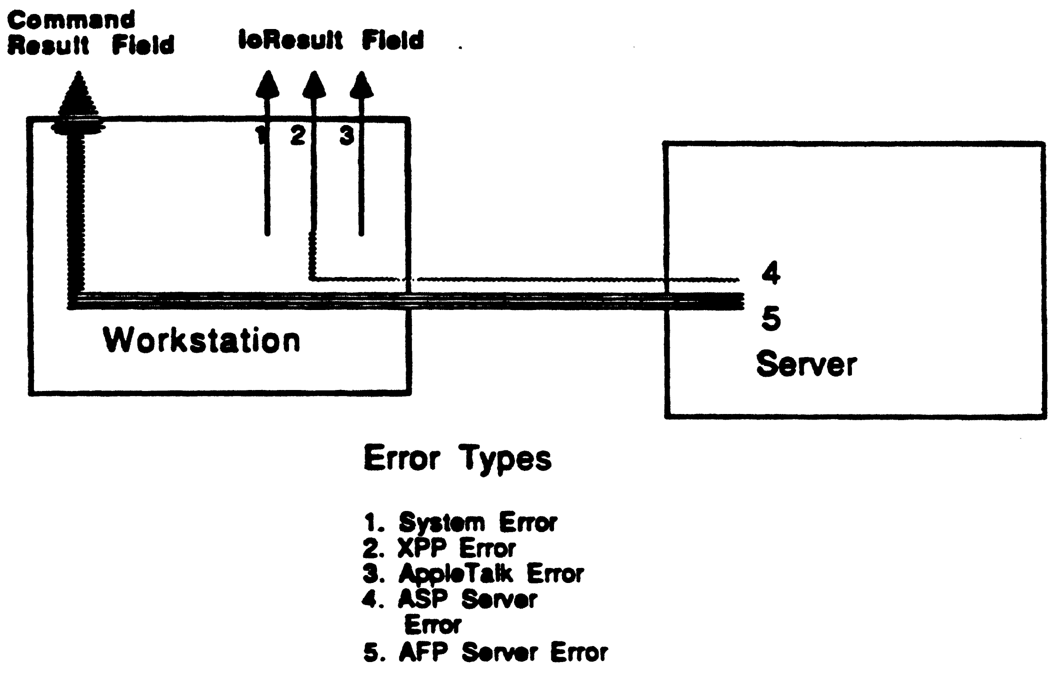

Figure 3. Error Reporting

AppleTalk Filing Protocol errors

In addition, the AppleTalk Filing Protocol defines errors that are returned from the server to the workstation client. These errors are returned in the the *CmdResult* field of the parameter block (error type 5 in Figure 3). This field is valid if no system-level error is returned by the call. Note that at the ASP level, the *CmdResult* field is client-defined data and may not be an error code.

.XPP driver functions overview

The paragraphs below describe the implementation of ASP in the .XPP driver. For more detailed information about ASP, refer to *Inside AppleTalk*, Section 11, "AppleTalk Session Protocol (ASP)."

Using AppleTalk Name Binding Protocol

A server wishing to advertise its service on the AppleTalk network calls ATP to open an ATP responding socket known as the *session listening socket* (SLS). The server then calls the Name Binding Protocol (NBP) to register a name on this socket. At this point, the server calls the server side of ASP to pass it the address of the SLS. Then, the server starts listening on the SLS for session opening requests coming over the network.

Opening and closing sessions

When a workstation wishes to access a server, the workstation must call NBP to discover the SLS for that server. Then the workstation calls ASP to open a session with that server.

After determining the SLS (address) of the server, the workstation client issues an OpenSession (or AFPLogin) call to open a session with that server. As a result of this call, ASP sends a special OpenSession packet (an ATP request) to the SLS; this packet carries the address of a workstation socket for use in the session. This socket is referred to as the workstation session socket (WSS). If the server is unable to set up the session, it returns an error. If the request is successful, the server returns no error and the session is opened. The open session packet also contains a version number so that both ends can verify that they are speaking the same version of ASP.

The AbortOS function can be used to abort an outstanding OpenSession request before it has completed.

The workstation client closes the session by issuing a CloseSession (or AFPLogout). The CloseSession call aborts any calls that are active on the session and closes the session. The session can also be closed by the server or by ASP itself, such as when one end of the session dies. The CloseAll call (which should be used with care) aborts every session that the driver has active.

Session maintenance

A session will remain open until it is explicitly terminated by the ASP client at either end or until one of the sessions ends dies or becomes unreachable.

Commands on an open session

Once a session has been opened, the workstation client can send a sequence of commands over the session to the server end. The commands are delivered in the same order as they are issued from the workstation end, and replies to the commands are returned to the workstation end.

Three types of commands can be made on an open session. These commands are UserCommand, UserWrite, and AFPCall functions described in the following paragraphs.

UserCommand calls are similar to ATP requests. The workstation client sends a command (included in a variable size command block) to the server client requesting it to perform a particular function and send back a variable size command reply. Examples of such commands vary from a request to open a particular file on a file server, to reading a certain range of bytes from a device. In the first case, a small amount of reply data is returned, in the second case a multiple-packet reply might be generated.

The .XPP driver does not interpret the command block or in any way participate in the command's function. It simply conveys the command block, included in a higher-level format, to the server end of the session, and returns the command reply to the workstation-end client. The command reply consists of a four-byte command result and a variable size command reply block.

UserWrite allows the workstation to convey blocks of data to the server. UserWrite is used to transfer a variable size block of data to the server end of the session and to receive a reply.

The AFPCall function provides a mechanism for passing an AFP command to the server end of an open session and receiving a reply. The first byte of the AFPCall command buffer contains the code for the AFP command that is to be passed to the server for execution. Most AFP calls are implemented through a very simple one-to-one mapping that takes the call and makes an ASP command out of it.

The AFPCall function can have one of four different formats. These four formats, which are basically all very similar,are described in detail below.

Getting server status information

ASP provides a service to allow its workstation clients to obtain a block of service status information from a server without the need for opening a session. The *GetStatus* function returns a status block from the server identified by the indicated address. ASP does not impose any structure on the status block. This structure is defined by the protocol above ASP.

Attention mechanism

Attentions are defined in ASP as a way for the server to alert the workstation of some event or critical piece of information. The ASP OpenSession and AFP login calls include a pointer to an attention routine in their parameter blocks. This attention routine is called by the .XPP driver when it receives an attention from the server and also when the session is closing as described below.

In addition, upon receiving an OpenSession call, or AFPlogin call, the .XPP driver sets the first two bytes of the session control block (SCB) to zero. When the .XPP driver receives an attention, the first two bytes of the SCB are set to the attention bytes from the packet (which are always non-zero).

> *Note:* A higher-level language such as Pascal may not wish to have a low-level attention routine called. A Pascal program can poll the attention bytes, and if they are ever nonzero, the program will know that an attention has come in (it would then set the attention bytes back to zero). Of course, two or more attentions could be received between successive polls and only the last one will be recorded.

The .XPP driver also calls the attention routine when the session is closed by either the server, workstation, or ASP itself (that is, timeout). In these cases, the attention bytes in the SCB are unchanged.

The attention routine

The attention routine is called at interrupt level and must observe interrupt conventions. Specifically, the interrupt routine can change registers A0 through A3 and D0 through D3 and it must not make any Memory Manager calls.

It will be called with

- D0 (word) equal to the SessRefnum for that session (see OpenSession Function)
- D1 (word) equal to the attention bytes passed by the server (or zero if the session is closing)

Return with an RTS (return from subroutine) to resume normal execution.

The next section describes the calls that can be made to the .XPP driver.

CALLING THE XPP DRIVER

This section describes how to use the XPP driver and how to call the XPP driver routines from assembly language and Pascal.

Using XPP

The XPP driver implements the workstation side of ASP and provides a mechanism for the workstation to send AppleTalk Filing Protocol (AFP) commands to the server.

Allocating memory

Every call to the XPP driver requires the caller to pass in whatever memory is needed by the driver for the call, generally at the end of the queue element. When a session is opened, the memory required for maintenance of that session (i.e., the Session Control Block) is also passed in.

For standard Device Manager calls, a queue element of a specific size equal to IOQElSize is allocated. When issuing many calls to XPP, it is the caller's responsibility to allocate a queue element that is large enough to accommodate the XPP driver's requirements for executing that call, as defined below. Once allocated, that memory can't be modified until the call completes.

Opening the XPP driver

To open the XPP driver, issue a Device Manager Open call. Refer to *Inside Macintosh,* Volume 2, Chapter 6, "The Device Manager." The name of the XPP driver is '.XPP'. Note that original Macintosh ROMs require that XPP be opened only once. With new ROMs, the XPP unit number can always be obtained through an *Open* call. With old ROMs only, the XPP unit number must be hard coded to XPPUnitNum (40) since only one open call can be issued to the driver.

The XPP driver cannot be opened unless AppleTalk is open. The application must ensure that the MPP and ATP drivers are opened, as described in *Inside Macintosh* Volume 2, pages 304-305.

The XPPLoaded bit (bit 5) in the PortBUse byte in low memory indicates whether or not the .XPP driver is open.

Example

The following is an example of the procedure an application might use to open the .XPP driver.

```
;-----------------------------------------------------------------------
;
;       Routine: OpenXPP
;
;               Open the .XPP driver and return the driver refNum for it.
;
;               Exit:  D0 = error code (ccr's set)
;                      D1 = XPP driver refNum (if no errors)
;
;               All other registers preserved
;
xppUnitNum      EQU     40                      ; default XPP driver number
xppTfRNum       EQU     -(xppUnitNum+1)         ; default XPP driver refNum

OpenXPP
                MOVE.L A0-A1/D2,-(SP)           ; save registers
                MOVE   ROM85,D0                 ; Check ROM type byte
                BPL.S  @10                      ; branch if >=128K ROMs
                BTST   #xppLoadedBit,PortBUse; is the XPP driver open already?
                BEQ.S  @10                      ; if not open, then branch to Open code.
                MOVE   #xppTfRNum,D1            ; else use this as driver refnum.
                MOVEQ  #0,D0                    ; set noErr.
                BRA.S  @90                      ; and exit.
;
; XPP driver not open. Make an _Open call to it. If using a 128K ROM
; machine and the driver is already open, we will make another Open call to
; it just so we get the correct driver refNum.
;
@10             SUB    #ioQElSize,SP            ; allocate temporary param block.
                MOVE.L SP,A0                    ; A0 -> param block.
                LEA    XPPName, A1              ; A1 -> XPP (ASP/AFP) driver name.
                MOVE.L A1,ioFileName(A0)        ; driver name into param block.
                CLR.B  ioPermssn(A0)            ; clear permissions byte
                _Open
                MOVE   ioRefNum(A0),D1          ; D1=driver refNum (invalid if error)
                ADD    #ioQElSize,SP            ; deallocate temp param block.
@90             MOVE.L (SP)+,A0-A1/D2           ; restore registers
                TST    D0                       ; error? (set ccr's)
                RTS

XPPName         DC.B   4                        ; length of string.
                DC.B   '.XPP'                   ; driver name.
```

From Pascal, XPP can be opened through the OpenXPP call, which returns the driver's reference number.

```
FUNCTION OpenXPP (VAR XPPRefnum: INTEGER): OSErr;
```

Open errors

- Errors returned by System
- portInUse is returned if the AppleTalk port is in use by a driver other than AppleTalk or if AppleTalk is not open.

Closing the XPP driver

To close the XPP driver, call the Device Manager Close routine.

> *Caution*: There is generally no reason to close the driver. Use this call sparingly, if at all. This call should generally be used only by system level applications.

Close Errors

- Errors returned by System
- Closerr (new ROMs only) returned if you try to close the driver and there are sessions active through that driver. When sessions are active, closerr is returned and the driver remains open.
- On old ROMs the driver is closed whether or not sessions are active and no error is returned. Results are unpredictable if sessions are still active.

Session Control Block

The session control block (SCB) is a non-relocatable block of data passed by the caller to XPP upon session opening. XPP reserves this block for use in maintaining an open session. The SCB size is defined by the constant SCBMemSize. The SCB is a locked block and as long as the session is open, the SCB can not be modified in any way by the application. There is one SCB for each open session. This block can be reused once a CloseSess call is issued and completed for that session or when the session is indicated as closed.

How to access the XPP driver

This section contains information for programmers using Pascal and assembly language routines.

Most XPP driver routines can be executed either synchronously (meaning that the application can't continue until the routine is completed) or asynchronously (meaning that the application is free to perform other tasks while the routine is executing).

Using Pascal

On asynchronous calls, the caller may pass a completion routine pointer in the parameter block, at offset IOCompletion. This routine will be executed upon completion of the call. It is executed at interrupt level and must not make any memory manager calls. If it uses application globals, it must ensure that register A5 is set up correctly; for details see SetupA5 and RestoreA5 in the Operating System Utilities chapter of *Inside Macintosh*. If no completion routine is desired, IOCompletion should be set to NIL.

Asynchronous calls return control to the caller with result code of noErr as soon as they are queued to the driver. This isn't an indication of successful completion. To determine when the call is actually completed, if you don't want to use a completion routine, you can poll the ioResult field; this field is set to 1 when the call is made, and receives the actual result code upon completion.

As different XPP calls take different arguments in their parameter block, a Pascal variant record has been defined to account for all the different cases. This parameter block is detailed in figure 4. The first four fields (which are the same for all calls) are automatically filled in by the device manager. The CSCode field is automatically filled in by Pascal, depending on which call is being made. The caller must, however, set the ioRefnum field to XPP's reference number, as returned in the OpenXPP call. The ioVRefnum field is unused.

Note that the parameter block is defined so as to be the maximum size used by any call. Different calls take different size parameter blocks, each call requiring a certain minimum size. Callers are free to abbreviate the parameter block where appropriate.

General

With each routine, a list of the parameter block fields used by the call is also given. All routines are invoked by Device Manager Control calls with the csCode field equal to the code corresponding to the function being called. The number next to each field name indicates the byte offset of the field from the start of the parameter block pointed to by A0; only assembly-language programmers need to be concerned with it.. An arrow next to each parameter name indicates whether it's an input, output, or input/output parameter:

Arrow	**Meaning**
-->	Parameter is passed
<--	Parameter is returned
<-->	Parameter is passed and returned

All Device Manager Control calls return an integer result code in the ioResult field.Each routine description lists all the applicable result codes, along with a short description of what the result code means. Refer to the section *XPP Driver result codes* for an alphabetical list of result codes returned by the .XPP driver.

Each routine description includes a Pascal form of the call. Pascal calls to the XPP Driver are of the form:

```
FUNCTION XPPCall(ParamBlock:XPPParmBlkPtr,async:BOOLEAN): OSErr;
```

XPPCall is the name of the routine.

ParamBlock points to the actual I/O queue element used in the _Control call, filled in by the caller with the parameters of the routine.

async indicates whether or not the call should be made asynchronously. If async is TRUE, the call is executed asynchronously; otherwise the call is executed synchronously.

The routine returns an result code of type OSErr.

```
XPPParamBlock = PACKED RECORD
    qLink:          QElemPtr;          { next queue entry }
    qType:          INTEGER;           { queue type }
    ioTrap:         INTEGER;           { routine trap }
    ioCmdAddr:      Ptr;               { routine address }
    ioCompletion:   ProcPtr;           { completion routine }
    ioResult:       OSErr;             { result code }
    CmdResult:      LONGINT;           { Command result (ATP user bytes) [long] }
    ioVRefNum:      INTEGER;           { volume reference or drive number }
    ioRefNum:       INTEGER;           { driver reference number }
    csCode:         INTEGER;           { Call command code }
    CASE XPPPrmBlkType OF
      ASPAbortPrm:
         (AbortSCBPtr: Ptr);                     { SCB pointer for AbortOS [long] }
     ASPSizeBlk:
         (ASPMaxCmdSize:  INTEGER;               { For SPGetParms [word] }
         ASPQuantumSize:  INTEGER;               { For SPGetParms [word] }
         NumSesss:        INTEGER);              { For SPGetParms [word] }
     XPPPrmBlk:
         (SessRefnum:     INTEGER;               { Offset to session refnum [word] }
         ASPTimeout:      Byte;                  { Timeout for ATP [byte] }
         ASPRetry:        Byte;                  { Retry count for ATP [byte] }
         CASE XPPSubPrmType OF
           ASPOpenPrm:
                 (ServerAddr:  LONGINT;          { Server address block [longword] }
                 SCBPointer:   Ptr;              { SCB pointer [longword] }
                 AttnRoutine:  Ptr);             { Attention routine pointer [long] }
           ASPSubPrm:
                 (CBSize:      INTEGER;          { Command block size [word] }
                 CBPtr:        Ptr;              { Command block pointer [long] }
                 RBSize:       INTEGER;          { Reply buffer size [word] }
                 RBPtr:        Ptr;              { Reply buffer pointer [long] }
                 CASE XPPEndPrmType OF
                   AFPLoginPrm:
                     (AFPAddrBlock: LONGINT; { Address block in AFP login [long] }
                     AFPSCBPtr:        Ptr;     { SCB pointer in AFP login [long] }
                     AFPAttnRoutine: Ptr);      { Attn routine pointer in AFP login }
                   ASPEndPrm:
                     (WDSize: INTEGER;         { Write data size [word] }
                     WDPtr:  Ptr;              { Write data pointer [long] }
                     CCBStart:  ARRAY[0..295] OF Byte))); {CCB memory for driver}
                     {Write max size(CCB) = 296; all other calls = 150}
    END;
```

Figure 4. .XPP Driver Parameter Block Record

AppleTalk Session Protocol Changes

This section contains descriptions of the XPP driver functions that you can call. Each function description shows the required parameter block fields, their offsets within the parameter block and a brief definition of the field. Possible result codes are also described.

Note on result codes

An important distinction exists between the ASPParamErr and ASPSessClosed result codes that may be returned by the XPP driver.

When the driver returns ASPParamErr to a call that takes as an input a session reference number, the session reference number does not relate to a valid open session. There could be several reasons for this, such as the workstation or server end closed the session or the server end of the session died.

The ASPSessClosed result code indicates that even though the session reference number relates to a valid session, that particular session is in the process of closing down (although the session is not yet closed).

OpenSess function

OpenSess initiates (opens) a session between the workstation and a server. Below is the required parameter block. A brief definition of the fields follows.

```
FUNCTION ASPOpenSession(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

—>	26	csCode	word	;always OpenSess
<—	28	SessRefnum	word	;session reference number
—>	30	ASPTimeout	byte	;retry interval in seconds
—>	31	ASPRetry	byte	;number of retries
—>	32	ServerAddr	long word	;server socket address
—>	36	SCBPointer	pointer	;pointer to session control block
—>	40	AttnRoutine	pointer	;pointer to attention routine

SessRefnum is a unique number identifying the open session between the workstation and the server. The SessRefnum is returned when the function completes successfully and is used in all calls to identify the session.

ASPTimeOut is the interval in seconds between retries of the open session request.

ASPRetry is the number of retries that will be attempted.

ServerAddr is the network identifier or address of the socket on which the server is listening.

SCBpointer points to a non-relocatable block of data for the session control block (SCB) that the XPP driver reserves for use in maintaining an open session. The SCB size is defined by the constant SCBMemSize. The SCB is a locked block and as long as the session is open, the SCB cannot be modified in any way by the application. There is one SCB for each open session. This block can be reused when a CloseSess call is issued and completed for that session, or when the session is indicated as closed through return of an ASPParamErr as the result of a call for that session.

AttnRoutine is a pointer to a routine that is invoked if an attention from the server is received, or upon session closing. If this pointer is equal to zero, no attention routine will be invoked.

Result codes	ASPNoMoreSess	Driver cannot support another session
	ASPParamErr	Server returned bad (positive) error code
	ASPNoServers	No servers at that address
		The server did not respond to the request.
	ReqAborted	OpenSess was aborted by an AbortOS
	ASPBadVersNum	Server cannot support the offered version number
	ASPServerBusy	Server cannot open another session

Note: The number of sessions that the driver is capable of supporting depends on the machine that the driver is running on.

CloseSess function

CloseSess closes the session identified by the SessRefnum returned in the OpenSess call. CloseSess aborts any calls that are active on the session, closes the session, and calls the attention routine, if any, with an attention code of zero (zero is invalid as a real attention code).

```
FUNCTION ASPCloseSession(xParamBlock:XPPParmBlkPtr;async:BOOLEAN):
      OSErr;
```

Parameter block

—>	26	csCode	word	;always CloseSess
—>	28	SessRefnum	word	;session reference number

Result codes	ASPParamErr	Parameter error, indicates an invalid session reference number
	ASPSessClosed	Session already in process of closing

AbortOS function

AbortOS aborts a pending (not yet completed) OpenSess call. The aborted OpenSess call will return a ReqAborted error.

```
FUNCTION ASPAbortOS(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

—>	26	csCode	word	;always AbortOS
—>	28	AbortSCBPointer	pointer	;pointer to session control block

AbortSCBPointer points to the original SCB used in the the pending OpenSess call.

Result codes	cbNotFound	SCB not found, no outstanding open session to be aborted. Pointer did not point to an open session SCB.

GetParms function

GetParms returns three ASP parameters. This call does not require an open session.

```
FUNCTION ASPGetParms(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

—>	26	csCode	word	;always GetParms
<—	28	ASPMaxCmdSize	word	;maximum size of command block
<—	30	ASPQuantumSize	word	;maximum data size
<—	32	NumSesss	word	;number of sessions

ASPMaxCmdSize is the maximum size of a command that can be sent to the server.

ASPQuantumSize is the maximum size of data that can be transferred to the server in a Write request or from the server in a command reply.

NumSesss is the number of concurrent sessions supported by the driver.

CloseAll function

CloseAll closes every session that the driver has active, aborting all active requests and invoking the attention routines where provided. This call should be used carefully. CloseAll can be used as a system level resource for making sure all sessions are closed prior to closing the driver.

```
FUNCTION ASPCloseAll(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

—>	26	csCode	word	;always CloseAll

UserWrite function

UserWrite transfers data on a session. UserWrite is one of the two main calls that can be used to transfer data on an ASP session. The other call that performs a similar data transfer is UserCommand described below. The UserWrite command returns data in two different places. Four bytes of data are returned in the CmdResult field and a variable size reply buffer is also returned.

```
FUNCTION ASPUserWrite(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

<—	18	CmdResult	long word	;ASP command result
—>	26	csCode	word	;always UserWrite
—>	28	SessRefnum	word	;session reference number
—>	30	ASPTimeout	byte	;retry interval in seconds

--->	32	CBSize	word	;command block size
--->	34	CBPtr	pointer	;command block pointer
<-->	38	RBSize	word	;reply buffer size and reply size
--->	40	RBPointer	pointer	;reply buffer pointer
<-->	44	WDSize	word	;write data size
--->	46	WDPtr	pointer	;write data pointer
<---	50	CCBStart	record	;start of memory for CCB

CmdResult is four bytes of data returned by the server.

SessRefnum is the session reference number returned in the OpenSess call.

ASPTimeOut is the interval in seconds between retries of the call. Notice that there is no ASPRetry field (retries are infinite). The command will be retried at the prescribed interval until completion or the session is closed.

CBSize is the size in bytes of the command data that is to be written on the session. The size of the command block must not exceed the value of ASPMaxCmdSize returned by the GetParms call. Note that this buffer is not the data to be written by the command but only the data of the command itself.

CBPointer points to the command data.

RBSize is passed and indicates the size of the reply buffer in bytes expected by the command. RBSize is also returned and indicates the size of the reply that was acutally returned.

RBPointer points to the reply buffer.

WDSize is passed and indicates the size of the write data in bytes to be sent by the command. WDSize is also returned and indicates the size of the write data that was acutally written.

WDPointer points to the write data buffer.

CCBStart is the start of the memory to be used by the XPP driver for the command control block. The size of this block is equal to a maximum of 296 bytes. To determine the exact requirement, refer to the CCB Sizes section of this document.

Result codes	ASPParam	Invalid session number, session has been closed
	ASPSizeErr	Command block size is bigger than MaxCmdSize
	ASPSessClosed	Session is closing
	ASPBufTooSmall	Reply is bigger than response buffer Buffer will be filled, data will be truncated

UserCommand function

UserCommand is used to send a command to the server on a session.

```
FUNCTION ASPUserCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

<---	18	CmdResult	long word	;ASP command result
--->	26	csCode	word	;always UserCommand
--->	28	SessRefnum	word	;session number
--->	30	ASPTimeout	byte	;retry interval in seconds
--->	32	CBSize	word	;command block size

—>	34	CBPtr	pointer	;command block pointer
<—>	38	RBSize	word	;reply buffer and reply size
—>	40	RBPointer	pointer	;reply buffer pointer
<—	50	CCBStart	record	;start of memory for CCB

SessRefnum is the session reference number returned in the OpenSess call.

ASPTimeOut is the interval in seconds between retries of the call. Notice that there is no ASPRetry field (retries are infinite). The command will be retried at the prescribed interval until completion or the session is closed.

CBSize is the size in bytes of the block of data that contains the command to be sent to the server on the session. The size of the command block must not exceed the value of ASPMaxCmdSize returned by the GetParms call.

CBPointer points to the block of data containing the command that is to be sent to the server on the session.

RBSize is passed and indicates the size of the reply buffer in bytes expected by the command. RBSize is also returned and indicates the size of the reply that was actually returned.

RBPointer points to the reply buffer.

CCBStart is the start of the memory to be used by the XPP driver for the command control block. The size of this block is equal to a maximum of 150 bytes. To determine the exact requirement refer to the CCB Sizes section of this document.

Result codes	ASPParamErr	Invalid session number, session has been closed
	ASPSizeErr	Command block size is bigger than MaxCmdSize
	ASPSessClosed	Session is closing
	ASPBufTooSmall	Reply is bigger than response buffer The buffer will be filled, data will be truncated

GetStatus function

GetStatus returns server status. This call is also used as GetServerInfo at the AFP level. This call is unique in that it transfers data over the network without having a session open. This call does not pass any data but requests that server status be returned.

```
FUNCTION ASPGetStatus(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

—>	26	csCode	word	;always GetStatus
—>	30	ASPTimeout	byte	;retry interval in seconds
—>	31	ASPRetry	byte	;number of retries
—>	32	ServerAddr	long word	;server socket address
<—>	38	RBSize	word	;reply buffer and reply size
—>	40	RBPointer	pointer	;reply buffer pointer
<—	50	CCBStart	record	;start of memory for CCB

ASPTimeOut is the interval in seconds between retries of the call.

ASPRetry is the number of retries that will be attempted.

ServerAddr is the network identifier or address of the socket on which the server is listening.

RBSize is passed and indicates the size of the reply buffer in bytes expected by the command. RBSize is also returned and indicates the size of the reply that was actually returned.

RBPointer points to the reply buffer.

CCBStart is the start of the memory to be used by the XPP driver for the command control block. The size of this block is equal to a maximum of 200 bytes. To determine the exact requirement refer to the CCB Sizes section of this document.

Result codes	ASPBufTooSmall	Reply is bigger than response buffer Replysize is bigger than ReplyBuffsize.
	ASPNoServer	No response from server at address used in call

AFP implementation

The AFPCall (AFPCommand) function passes a command to an AFP server. The first byte of the AFPCall command buffer (the AFP command byte) must contain a valid AFP command code.

> Note: Server information should be gotten through a GetStatus call (described above). GetStatus is equivalent to the AFPGetSrvrInfo. Making an AFP GetSrvrInfo call using AFPCommand, results in a error.

Mapping AFP commands

Most AFP calls are implemented by XPP through a very simple one-to-one mapping of an AFP call to an ASP call without interpretation or verification of the data.

The XPP driver maps AFP command codes to ASP commands according to the following conventions:

AFP Command Code	Comment
$00	Invalid AFP command
$01–$BE (1–190)	Mapped to UserCommand (with the exceptions listed below)
$BF (191)	Mapped to UserCommand . Reserved for developers; will never be used by Apple
$C0–$FD (192–253)	Mapped to UserWrite
$FE (254)	Mapped to UserWrite; will never be used by Apple
$FF (255)	Invalid AFP command

The following AFP calls are exceptions to the above conventions:

AFP Command (Code/decimal)	Comment
GetSrvrInfo (15)	Mapped to GetStatus (Use GetStatus to make this call)
Login (18)	Mapped to appropriate login dialog including OpenSess call
LoginCont (19)	Mapped to appropriate login dialog

Logout (20)	Mapped to CloseSess
Write (33)	Mapped to UserWrite

Note that the following AFP calls can pass or return more data than can fit in QuantumSize bytes (8 ATP response packets) and may be broken up by XPP into multiple ASP calls.

AFP Command (Code/decimal)	**Comment**
Read (27)	Can return up to the number of bytes indicated in ReqCount
Write (33)	Can pass up to the number of bytes indicated in ReqCount

AFPCall function

The AFPCall function can have one of the following command formats.

- General
- Login
- AFPWrite
- AFPRead

General command format

The general command format for the AFPCall function passes an AFP command to the server. This format is used for all AFP calls except AFPLogin, AFPRead and AFPWrite. Note that from Pascal this call is know as AFPCommand.

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

<--	18	CmdResult	long word	;AFP command result
-->	26	csCode	word	;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 reply size
-->	40	RBPointer	pointer	;reply buffer pointer
<-->	44	WDSize	word	;write data size
-->	46	WDPtr	pointer	;write data pointer
<--	50	CCBStart	record	;start of memory for CCB

CmdResult is four bytes of data returned from the server containing an indication of the result of the AFP command.

SessRefnum is the session reference number returned in the AFPLogin call.

ASPTimeOut is the interval in seconds between retries of the call by the driver.

CBSize is the size in bytes of the block of data that contains the command to be sent to the server on the session. The size of the command block must not exceed the value of ASPMaxCmdSize returned by the GetParms call.

CBPointer points to start of the block of data (command block) containing the command that is to be sent to the server on the session. The first byte of the command block must contain the AFP command byte. Subsequent bytes in the command buffer contain the parameters assoicated with the command as defined in the AFP document.

RBSize is passed and indicates the size of the reply buffer in bytes expected by the command. RBSize is also returned and indicates the size of the reply that was acutally returned.

RBPointer points to the reply buffer.

WDSize is the size of data to be written to the server (only used if the command is one that is mapped to an ASP UserWrite).

WDPointer points to the write data buffer (only used if the command is one that is mapped to an ASP UserWrite).

CCBStart is the start of the memory to be used by the XPP driver for the command control block. The size of this block is equal to a maximum of 296 bytes. To determine the exact requirement refer to the CCB Sizes section of this document.

Result codes	ASPParamErr	Invalid session number; session has been closed
	ASPSizeErr	Command block size is bigger than MaxCmdSize
	ASPSessClosed	Session is closing
	ASPBufTooSmall	Reply is bigger than response buffer Buffer will be filled, data will be truncated
	ParmError	AFP command block size is equal to zero. This error will also be returned if the command byte in the command block is equal to 0 or $FF (255) or GetSrvrStatus (15).

Login command format

The AFP login command executes a series of AFP operations as defined in the AFP Draft Proposal. For further information, refer to the AFP document.

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

<--	18	CmdResult	long word	;AFP command result
-->	26	csCode	word	;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 reply size
-->	40	RBPtr	pointer	;reply buffer pointer
-->	44	AFPAddrBlock	long word	;server address block
<-->	48	AFPSCBPointer	pointer	;SCB pointer
<-->	52	AFPAttnRoutine	pointer	;attention routine pointer
<--	50	CCBStart	record	;start of command control block

CmdResult is four bytes of data returned from the server containing an indication of the result of the AFP command.

SessRefnum is the session reference number (returned by the AFPLogin call).

ASPTimeOut is the interval in seconds between retries of the call.

ASPRetry is the number of retries that will be attempted.

CBSize is the size in bytes of the block data that contains the command to be sent to the server on the session. The size of the command block must not exceed the value of ASPMaxCmdSize returned by the GetParms call.

CBPointer points to the block of data (command block) containing the AFP login command that is to be sent to the server on the session. The first byte of the command block must be the AFP login command byte. Subsequent bytes in the command buffer contain the parameters associated with the command.

RBSize is passed and indicates the size of the reply buffer in bytes expected by the command. RBSize is also returned and indicates the size of the reply that was acutally returned.

RBPointer points to the reply buffer.

AFPServerAddr is the network identifier or address of the socket on which the server is listening.

AFPSCBPointer points to a locked block of data for the session control block (SCB).The SCB size is defined by SCBMemSize. The SCB is a locked block and as long as the session is open, the SCB can not be modified in any way by the application. There is one SCB for each open session.

AFPAttnRoutine is a pointer to a routine that is invoked if an attention from the server is received. When AFPAttnRoutine is equal to zero, no attention routine will be invoked.

CCBStart is the start of the memory to be used by the XPP driver for the command control block. The size of this block is equal to a maximum of 150 bytes. To determine the exact requirement refer to the CCB Sizes section of this document.

Note : In the parameter block, the AFPSCBPointer and the AFPAttnRoutine fields overlap with the start of the CCB and are modified by the call.

Result codes	ASPSizeErr	Command block size is bigger than MaxCmdSize
	ASPBufTooSmall	Reply is bigger than response buffer Buffer will be filled, data will be truncated
	ASPNoServer	Server not responding
	ASPServerBusy	Server cannot open another session
	ASPBadVersNum	Server cannot support the offered ASP version number
	ASPNoMoreSess	Driver cannot support another session.

AFPWrite command format

The AFPWrite and AFPRead command formats allow the calling application to make AFP-level calls that read or write a data block that is larger than a single ASP-level call is capable of reading or writing. The maximum number of bytes of data that can be read or written at the ASP level is equal to QuantumSize.

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

<---	18	CmdResult	long word	;AFP command result
--->	26	csCode	word	;always AFPCall
--->	28	SessRefnum	word	;session 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 reply size
--->	40	RBPtr	pointer	;reply buffer pointer
<---	44	WDSize	word	;(used internally)
<-->	46	WDPtr	pointer	;write data pointer (updated)
<---	50	CCBStart	record	;start of memory for CCB

CmdResult is four bytes of data returned from the server containing an indication of the result of the AFP command.

SessRefnum is the session reference number returned in the AFPLogin call.

ASPTimeOut is the interval in seconds between retries of the call.

CBSize is the size in bytes of the block data that contains the command to be sent to the server on the session. The size of the command block must not exceed the value of ASPMaxCmdSize returned by the ASPGetParms call.

CBPointer points to the block of data (see command block structure below) containing the AFP write command that is to be sent to the server on the session. The first byte of the Command Block must contain the AFP write command byte.

RBSize is passed and indicates the size of the reply buffer in bytes expected by the command. RBSize is also returned and indicates the size of the reply that was acutally returned.

RBPointer points to the reply buffer.

WDSize is used internally.

> *Note:* This command does not pass the write data size in the queue element but in the command buffer. XPP will look for the size in that buffer.

WDPtr is a pointer to the block of data to be written. Note that this field will be updated by XPP as it proceeds and will always point to that section of the data which XPP is currently writing.

CCBStart is the start of the memory to be used by the XPP driver for the command control block. The size of this block is equal to a maximum of 296 bytes. To determine the exact requirement refer to the CCB Sizes section of this document.

Command block structure

The AFP write command passes several arguments to XPP in the command buffer itself. The byte offsets are relative to the location pointed to by *CBPointer.*

--->	0	CmdByte	byte	;AFP call command byte
--->	1	StartEndFlag	byte	;Start/end Flag
<-->	4	RWOffset	long word	;offset within fork to write
<-->	8	ReqCount	long word	;requested count

CmdByte is the AFP call command byte and must contain the AFP write command code.

StartEndFlag is a one-bit flag (the high bit of the byte) indicating whether the RWOffset field is relative to the beginning or the end of the fork (all other bits are zero).

0 = relative to the beginning of the fork

1 = relative to the end of the fork

RWOffset is the byte offset within the fork at which the write is to begin.

ReqCount indicates the size of the data to be written and is returned as the actual size written.

Note that the RWOffset and ReqCount fields are modified by XPP as the write proceeds and will always indicate the current value of these fields.

The Pascal structure of the AFP command buffer follows:

```
AFPCommandBlock = PACKED RECORD
        CmdByte:            Byte;
        StartEndFlag:       Byte;
        ForkRefNum:         INTEGER;      {Used by server}
        RWOffset:           LONGINT;
        ReqCount:           LONGINT;
        NewLineFlag:        Byte;         {Unused by write}
        NewLineChar:        CHAR;         {Unused by write}
END;
```

Result codes	ASPParamErr	Invalid session number
	ASPSizeErr	Command block size is bigger than MaxCmdSize
	ASPSessClosed	Session is closing
	ASPBufTooSmall	Reply is bigger than response buffer

AFPRead command format

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN): OSErr;
```

Parameter block

<--	18	CmdResult	long word	;ASP command result
-->	26	csCode	word	;always AFPCall
-->	28	SessRefnum	word	;session 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	;start of memory for CCB

CmdResult is four bytes of data returned from the server containing an indication of the result of the AFP command.

SessRefnum is the session reference number returned in the AFPLogin call.

ASPTimeOut is the interval in seconds between retries of the call.

CBSize is the size in bytes of the block data that contains the command to be sent to the server on the session. The size of the command block must not exceed the value of ASPMaxCmdSize returned by the GetParms call.

CBPointer points to the block of data (command block) containing the AFP read command that is to be sent to the server on the session. The first byte of the command block must contain the AFP read command byte. The command block structure is shown below.

RBSize is used internally.

> Note: This command does not pass the read size in the queue element but in the command buffer. XPP will look for the size in that buffer.

RBPointer points to the reply buffer. Note that this field will be updated by XPP as it proceeds and will always point to that part of the buffer that XPP is currently reading into.

CCBStart is the start of the memory to be used by the XPP driver for the command control block. The size of this block is equal to a maximum of 150 bytes. To determine the exact requirement refer to The CCB Sizes section .

Command block structure

The AFP read command passes several arguments to XPP in the command buffer itself. The byte offsets are relative to the location pointed to by CBPointer.

--->	0	CmdByte	byte	;AFP call command byte
<-->	4	RWOffset	long word	;offset within fork to read
<-->	8	ReqCount	long word	;requested count
--->	12	NewLineFlag	byte	;Newline Flag
--->	13	NewLineChar	byte	;Newline Character

CmdByte is the AFP call command byte and must contain the AFP read command code.

RWOffset is the byte offset within the fork at which the read is to begin.

ReqCount indicates the size of the read data buffer and is returned as the actual size read.

Note that the RWOffset and ReqCount fields are modified by XPP as the read proceeds and will always indicate the current value of these fields.

NewLineFlag is a one-bit flag (the high bit of the byte) indicating whether or not the read is to terminate at a specified character (all other bits are zero).

> 0 = no Newline Character is specified
>
> 1 = a Newline Character is specified

NewLineChar is any character from $00 to $FF (inclusive) which when encountered in reading the fork, causes the read operation to terminate.

The Pascal structure of the AFP command buffer follows:

```
AFPCommandBlock = PACKED RECORD
      CmdByte:          Byte;
      StartEndFlag:     Byte;        {Unused for read}
      ForkRefNum:       INTEGER;     {Used by server}
      RWOffset:         LONGINT;
      ReqCount:         LONGINT;
      NewLineFlag:      Byte;
      NewLineChar:      CHAR;
END;
```

Result codes	ASPParamErr	Invalid session number
	ASPSizeErr	Command block size is bigger than MaxCmdSize
	ASPSessClosed	Session is closing
	ASPBufTooSmall	Reply is bigger than response buffer

CCB sizes

The XPP driver uses the memory provided at the end of the UserWrite, UserCommand and GetStatus parameter blocks as an internal command control block (CCB). Using the maximum block sizes specified in the call descriptions will provide adequate space for the call to execute successfully. However, this section is provided for developers who wish to minimize the amount of memory taken up by the CCB in the queue element.

Specifically, this memory is used for creating data structures to be used in making calls to the ATP driver. This includes parameter blocks and buffer data structures (BDS's – detailed in the AppleTalk Manager chapter of *Inside Macintosh*). The exact size of this memory depends on the size of the response expected, and, in the case of UserWrite, on the size of data to be written.

In the UserCommand and GetStatus cases (along with all AFP calls which map to UserCommand), a BDS must be set up to hold the response information. The number of entries in this BDS is equal to the size of the response buffer divided by the maximum number of data bytes per ATP response packet (578), rounded up. Note that as described in the ASP document in *Inside AppleTalk*, ASP must ask for an extra response in the case where the response buffer is an exact multiple of 578. Of course, no BDS can be more than eight elements big. XPP also needs bytes for the queue element to call ATP with, so the minimum size of a CCB, as a function of the response buffer size (RBSize) is:

```
BDSSize = MIN (((RBSize DIV 578) + 1),8) * BDSEntrySz
CCBSize = IOQElSize + 4 + BDSSize
```

In the UserWrite (and AFP calls mapping to UserWrite) case, XPP needs to create an additional BDS and queue element to use in sending the write data to the server. Thus the minimum size of a UserWrite CCB, as a function of the response buffer and write data sizes (RBSize and WDSize) is:

```
WrBDSSize = MIN (((WDSize DIV 578) + 1),8) * BDSEntrySz
WrCCBSize = (2 * IOQElSize) + 4 + BDSSize + WrBDSSize
```

Note: BDSEntrySz is equal to 12. IOQElSize is equal to 50.

XPP Driver Result Codes

Result Code	Comment	Returned by
ASPBadVersNum	Server cannot support the offered version number	OpenSess AFPCall (Login)
ASPBufTooSmall	Reply is bigger than response buffer Buffer will be filled, data may be truncated.	UserWrite UserCommand GetStatus AFPCall
ASPNoMoreSess	Driver cannot support another session	OpenSess AFPCall (Login)
ASPNoServers	No servers at that address The server did not respond to the request.	GetStatus OpenSess AFPCall (Login)
ASPParamErr	Parameter error server returned bad (positive) error code Invalid Session Reference Number	OpenSess CloseSess UserWrite UserCommand AFPCall
ASPServerBusy	Server cannot open another session	OpenSess AFPCall (Login)
ASPSessClosed	Session already in process of closing	CloseSess UserWrite UserCommand AFPCall
ASPSizeErr	Command block size is bigger than MaxParamSize	UserWrite UserCommand AFPCall
cbNotFound	SCB not found, no outstanding open session to be aborted. Pointer did not point to an open session SCB.	AbortOS
ParmError	AFP Command Block size is less than or equal to zero. Command byte in the Command block is equal to 0 or $FF (255) or GetSrvrStatus (15).	AFPCall
ReqAborted	Open session was aborted by an Abort Open Session	OpenSess AFPCall (Login)

SUMMARY

SetSelfSend function

Parameter Block

--->	26	csCode	word	; always SetSelfSend
--->	28	NewSelfFlag	byte	; new SelfSend flag
<---	29	OldSelfFlag	byte	; old SelfSend flag

AppleTalk Transaction Protocol

NSendRequest function

Parameter block

--->	18	UserData	longword	;user bytes
<---	22	reqTID	word	;transaction ID used in requet
--->	26	csCode	word	;always sendRequest
--->	28	atpSocket	byte	;socket to send request on
<-->	29	atpFlags	byte	;control information
--->	30	addrBlock	longword	;destination socket address
--->	34	reqLength	word	;request size in bytes
--->	36	reqPointer	pointer	;pointer to request data
--->	40	bdsPointer	pointer	;pointer to response BDS
--->	44	numOfBuffs	byte	;number of responses expected
--->	45	timeOutVal	byte	;timeout interval
<---	46	numOf Resps	byte	;number of responses received
<-->	47	retryCount	byte	;number of retries
<---	48	intBuff	word	;used internally

KillSendReq function

Parameter block

--->	26	csCode	word	; always KillSendReq
--->	44	AKillQEl	pointer	; pointer to queue element

KillGetReq function

Parameter block

--->	26	csCode	word	; always KillGetReq
--->	44	AKillQEl	pointer	; pointer to queue element

Name Binding Protocol

KillNBP function

Parameter block

```
    -->  26   csCode     word      ; always KillNBP
    -->  28   NKillQEl   pointer   ; pointer to queue element
```

Data Types

```
XPPParamBlock = PACKED RECORD
    qLink:          QElemPtr;                  { next queue entry }
    qType:          INTEGER;                   { queue type }
    ioTrap:         INTEGER;                   { routine trap }
    ioCmdAddr:      Ptr;                       { routine address }
    ioCompletion:   ProcPtr;                   { completion routine }
    ioResult:       OSErr;                     { result code }
    CmdResult:      LONGINT;                   { Command result(ATP user bytes)[long]}
    ioVRefNum:      INTEGER;                   { volume reference or drive number }
    ioRefNum:       INTEGER;                   { driver reference number }
    csCode:         INTEGER;                   { Call command code }
    CASE XPPPrmBlkType OF
      ASPAbortPrm:
          (AbortSCBPtr: Ptr);                  { SCB pointer for AbortOS [long] }
     ASPSizeBlk:
          (ASPMaxCmdSize:  INTEGER;            { For SPGetParms [word] }
          ASPQuantumSize:  INTEGER;            { For SPGetParms [word] }
          NumSesss:        INTEGER);           { For SPGetParms [word] }
     XPPPrmBlk:
          (SessRefnum:     INTEGER;            { Offset to session refnum [word] }
          ASPTimeout:      Byte;               { Timeout for ATP [byte] }
          ASPRetry:        Byte;               { Retry count for ATP [byte] }
          CASE XPPSubPrmType OF
            ASPOpenPrm:
                  (ServerAddr:  LONGINT;       { Server address block [longword] }
                  SCBPointer:   Ptr;           { SCB pointer [longword] }
                  AttnRoutine:  Ptr);          { Attention routine pointer [long] }
            ASPSubPrm:
                  (CBSize:      INTEGER;       { Command block size [word] }
                  CBPtr:        Ptr;           { Command block pointer [long] }
                  RBSize:       INTEGER;       { Reply buffer size [word] }
                  RBPtr:        Ptr;           { Reply buffer pointer [long] }
                  CASE XPPEndPrmType OF
                    AFPLoginPrm:
                      (AFPAddrBlock: LONGINT; { Address block in AFP login [long] }
                      AFPSCBPtr:      Ptr;     { SCB pointer in AFP login [long] }
                      AFPAttnRoutine: Ptr);    { Attn routine pointer in AFP login }
                    ASPEndPrm:
                      (WDSize: INTEGER;        { Write data size [word] }
                      WDPtr:  Ptr;             { Write data pointer [long] }
                      CCBStart:  ARRAY[0..295] OF Byte))); { CCB memory  for driver }
    END;
```

```
AFPCommandBlock = PACKED RECORD
   CmdByte:        Byte;
   StartEndFlag:   Byte;        {Unused for read}
   ForkRefNum:     INTEGER;     {Used by server}
   RWOffset:       LONGINT;
   ReqCount:       LONGINT;
   NewLineFlag:    Byte;        {Unused by write}
   NewLineChar:    CHAR;        {Unused by write}
END;
```

AppleTalk Session Protocol

```
FUNCTION ASPOpenSession(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

--->	26	csCode	word	;always OpenSess
<---	28	SessRefnum	word	;session reference number
--->	30	ASPTimeout	byte	;retry interval in seconds
--->	31	ASPRetry	byte	;number of retries
--->	32	ServerAddr	long word	;server socket address (SLS)
--->	36	SCBPointer	pointer	;pointer to session control block
--->	40	AttnRoutine	pointer	;pointer to attention routine

```
FUNCTION ASPCloseSession(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

--->	26	csCode	word	;always CloseSess
--->	28	SessRefnum	word	;session reference number

```
FUNCTION ASPAbortOS(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

--->	26	csCode	word	;always AbortOS
--->	28	AbortSCBPtr	pointer	;pointer to session control block

```
FUNCTION ASPGetParms(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

--->	26	csCode	word	;always GetParms
<---	28	ASPMaxCmdSize	word	;maximum size of command block
<---	30	ASPQuantumSize	word	;maximum data size
<---	32	NumSesss	word	;number of sessions

```
FUNCTION ASPCloseAll(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

--->	26	csCode	word	;always CloseAll

```
FUNCTION ASPUserWrite(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

<--	18	CmdResult	long word	;ASP command result
-->	26	csCode	word	;always UserWrite
-->	28	SessRefnum	word	;session reference number
-->	30	ASPTimeout	byte	;retry interval in seconds
-->	32	CBSize	word	;command block size
-->	34	CBPtr	pointer	;command block pointer
<-->	38	RBSize	word	;reply buffer size and reply size
-->	40	RBPtr	pointer	;reply buffer pointer
<-->	44	WDSize	word	;write data size
-->	46	WDPtr	pointer	;write data pointer
<--	50	CCBStart	record	;start of memory for CCB

```
FUNCTION ASPUserCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

<--	18	CmdResult	long word	;ASP command result
-->	26	csCode	word	;always UserCommand
-->	28	SessRefnum	word	;session number
-->	30	ASPTimeout	byte	;retry interval in seconds
-->	32	CBSize	word	;command block size
-->	34	CBPtr	pointer	;command block pointer
<-->	38	RBSize	word	;reply buffer and reply size
-->	40	RBPtr	pointer	;reply buffer pointer
<--	50	CCBStart	record	;start of memory for CCB

```
FUNCTION ASPGetStatus(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

-->	26	csCode	word	;always GetStatus
-->	30	ASPTimeout	byte	;retry interval in seconds
-->	31	ASPRetry	byte	;number of retries
-->	32	ServerAddr	long word	;server socket address
<-->	38	RBSize	word	;reply buffer and reply size
-->	40	RBPtr	pointer	;reply buffer pointer
<--	50	CCBStart	record	;start of memory for CCB

AppleTalk Filing Protocol

AFPCall Function
General command format

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

<--	18	CmdResult	long word	;AFP command result
-->	26	csCode	word	;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 reply size
-->	40	RBPtr	pointer	;reply buffer pointer
<-->	44	WDSize	word	;write data size
-->	46	WDPtr	pointer	;write data pointer
<--	50	CCBStart	record	;start of memory for CCB

Login command format

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

<--	18	CmdResult	long word	;AFP command result
-->	26	csCode	word	;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 reply size
-->	40	RBPtr	pointer	;reply buffer pointer
-->	44	AFPAddrBlock	long word	;server address block
<-->	48	AFPSCBPointer	pointer	;SCB pointer
<-->	52	AFPAttnRoutine	pointer	;attention routine pointer
<--	50	CCBStart	record	;start of command control block

AFPWrite command format

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

<--	18	CmdResult	long word	;AFP command result
-->	26	csCode	word	;always AFPCall
-->	28	SessRefnum	word	;session 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 reply size
--->	40	RBPtr	pointer	;reply buffer pointer
<---	44	WDSize	word	;(used internally)
<-->	46	WDPtr	pointer	;write data pointer (updated)
<---	50	CCBStart	record	;start of memory for CCB

Command block structure

--->	0	CmdByte	byte	;AFP call command byte
--->	1	StartEndFlag	byte	;Start/end Flag
<-->	4	RWOffset	long word	;offset within fork to write
<-->	8	ReqCount	long word	;requested count

AFPRead command format

```
FUNCTION AFPCommand(xParamBlock:XPPParmBlkPtr;async:BOOLEAN) OSErr;
```

Parameter block

<---	18	CmdResult	long word	;ASP command result
--->	26	csCode	word	;always AFPCall
--->	28	SessRefnum	word	;session 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	;start of memory for CCB

Command block structure

The AFP read command passes several arguments to XPP in the command buffer itself. The byte offsets are relative to the location pointed to by *CBPtr*.

--->	0	CmdByte	byte	;AFP call command byte
<-->	4	RWOffset	long word	;offset within fork to read
<-->	8	ReqCount	long word	;requested count
--->	12	NewLineFlag	byte	;Newline Flag
--->	13	NewLineChar	byte	;Newline Character

AppleTalk Session Protocol Constants

Offsets in user bytes

```
ASPCmdCode      EQU     0       ; Offset to command field
ASPWSSNum       EQU     1       ; WSS number in OpenSessions
ASPVersNum      EQU     2       ; ASP version number in OpenSessions
ASPSSSNum       EQU     0       ; SSS number in OpenSessReplies
ASPSessID       EQU     1       ; Session ID (requests &OpenSessReply)
ASPOpenErr      EQU     2       ; OpenSessReply error code

ASPSeqNum       EQU     2       ; Sequence number in requests
```

```
ASPAttnCode      EQU     2          ; Attention bytes in attentions
```

Offsets in ATP data part

```
ASPWrBSize       EQU     0       ;Offset to write buffer size (WriteData)
ASPWrHdrSz       EQU     ASPWrBSize+2 ; Size of data part
```

ASP command codes

```
ASPCloseSess     EQU     1               ; Close session
ASPCommand       EQU     2               ; User-command
ASPGetStat       EQU     3               ; Get status
ASPOpenSess      EQU     4               ; Open session
ASPTickle        EQU     5               ; Tickle
ASPWrite         EQU     6               ; Write
ASPDataWrite     EQU     7               ; WriteData (from server)
ASPAttention     EQU     8               ; Attention (from server)
```

ASP miscellaneous

```
ASPVersion       EQU     $0100           ; ASP version number
MaxCmdSize       EQU     ATPMaxData      ; Maximum command block size
QuantumSize      EQU     ATPMaxData*ATPMaxNum ; Maximum reply size
XPPLoadedBit     EQU     ATPLoadedBit+1  ; XPP bit in PortBUse
XPPUnitNum       EQU     40              ; Unit number for XPP (old ROMs)
```

ASP errors codes

```
ASPBadVersNum    EQU     -1066  ; Server cannot support this ASP version
ASPBufTooSmall   EQU     -1067  ; Buffer too small
ASPNoMoreSess    EQU     -1068  ; No more sessions on server
ASPNoServers     EQU     -1069  ; No servers at that address
ASPParamErr      EQU     -1070  ; Parameter error
ASPServerBusy    EQU     -1071  ; Server cannot open another session
ASPSessClosed    EQU     -1072  ; Session closed
ASPSizeErr       EQU     -1073  ; Command block too big
```

Control codes

```
OpenSess         EQU     255    ; Open session
CloseSess        EQU     254    ; Close session
UserCommand      EQU     253    ; User command
UserWrite        EQU     252    ; User write
GetStatus        EQU     251    ; Get status
AFPCall          EQU     250    ; AFP command (buffer has command code)
GetParms         EQU     249    ; Get parameters
AbortOS          EQU     248    ; Abort open session request
CloseAll         EQU     247    ; Close all open sessions
```

ASP queue element standard structure: arguments passed in the CSParam area

```
SessRefnum       EQU     CSParam       ; Offset to session refnum [word]
ASPTimeout       EQU     SessRefnum+2  ; Timeout for ATP [byte]
ASPRetry         EQU     ASPTimeout+1  ; Retry count for ATP [byte]
ServerAddr       EQU     ASPRetry+1    ; Server address block [longword]
```

```
SCBPointer       EQU     ServerAddr+4 ; SCB pointer [longword]
AttnRoutine      EQU     SCBPointer+4 ; Attention routine pointer [long]

CBSize           EQU     ASPRetry+1      ; Command block size [word]
CBPtr            EQU     CBSize+2        ; Command block pointer [long]
RBSize           EQU     CBPtr+4         ; Reply buffer size [word]
RBPtr            EQU     RBSize+2        ; Reply buffer pointer [long]
WDSize           EQU     RBPtr+4         ; Write data size [word]
WDPtr            EQU     WDSize+2        ; Write data pointer [long]
CCBStart         EQU     WDPtr+4         ; Start of memory for CCB

ASPMaxCmdSize    EQU     CSParam         ; For SPGetParms [word]
ASPQuantumSize   EQU     ASPMaxCmdSize+2 ; For SPGetParms [word]
AbortSCBPtr      EQU     CSParam         ; SCB pointer for AbortOS [long]

CmdResult        EQU     UserData ; Command result (ATP user bytes)[long]

AFPAddrBlock     EQU     RBPtr+4  ; Address block in AFP login[long]
AFPSCBPtr        EQU     AFPAddrBlock+4 ; SCB pointer in AFP login [long]
AFPAttnRoutine   EQU     AFPSCBPtr+4 ; Attn routine pointer in AFP login

SCBMemSize       EQU     $C0             ; Size of memory for SCB
```

AFPCall command codes

```
AFPLogin            EQU   18;
AFPContLogin        EQU   19;
AFPLogout           EQU   20;
AFPRead             EQU   27;
AFPWrite            EQU   33;
```

Offsets for certain parameters in Read/Write calls

```
StartEndFlag    EQU      1 ; Write only; offset relative to start or end
RWOffset        EQU      4 ; Offset at which to start read or write
ReqCount        EQU      8 ; Count of bytes to read or write
NewLineFlag     EQU      12 ; Read only; newline character flag
NewLineChar     EQU      13 ; Read only; newline character
LastWritten     EQU      0 ; Write only; last written  (returned)
```

Miscellaneous

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