This chapter describes the AppleTalk Session Protocol (ASP) that you can use to establish a session between an ASP workstation application or process and an ASP server application. An ASP session is asymmetrical: all communication is initiated by the ASP workstation and responded to by the ASP server.

ASP provides an application programming interface for the workstation side only. ASP is not commonly used by application program developers. The primary use of ASP is to provide services for the AppleTalk Filing Protocol (AFP) that, in turn, provides all of the services necessary to access an AppleTalk AppleShare server. Most developers who want to write an AppleTalk application that establishes a session use the AppleTalk Data Stream Protocol (ADSP) because it provides peer-to-peer services. For these reasons, this chapter includes “About” and “Reference” sections only; it does not include a “Using” section, as do most of the other chapters in this book. This chapter is included to complete the coverage of the AppleTalk protocol stack in this book.

However, if you want to use ASP to write an application that runs on a workstation and initiates a session with an ASP server, you should read this chapter and the chapter in *Inside AppleTalk*, second edition, that describes the AppleTalk Session Protocol specification.

You can use ASP to open and close a session with an ASP server; you can also send commands and data across the session to the server and receive replies in response. The commands that you send to the ASP server must adhere to the syntax of a higher-level protocol that is built on top of the ASP server. ASP transfers the commands; it does not interpret or execute them.

This chapter does not describe how to implement an ASP server. If you want to implement an ASP server, you must use the programming interface to the AppleTalk Transaction Protocol (ATP) and follow the AppleTalk Session Protocol specification as defined in *Inside AppleTalk*, second edition.

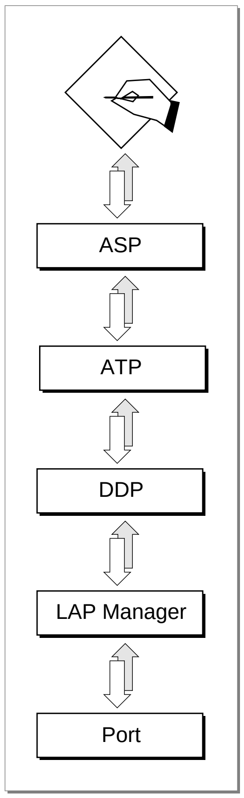
If you want to write an application that supports a peer-to-peer session in which each end of the session can send and receive data at any time, you should use the AppleTalk Data Stream Protocol (ADSP) instead of ASP. The chapter “AppleTalk Data Stream Protocol (ADSP)” in this book describes ADSP.

For an overview of ASP and how it fits within the AppleTalk protocol stack, read the chapter “Introduction to AppleTalk” in this book. “Introduction to AppleTalk” also introduces and defines some of the terminology used in this chapter. Because ASP is built on top of ATP, possessing an understanding of ATP will help you to understand ASP. The chapter “AppleTalk Transaction Protocol (ATP)” in this book describes ATP.

About ASP

The AppleTalk Session Protocol (ASP) allows one or more ASP workstation applications or processes to establish a session with the same server at the same time. To track communication from various sessions, ASP assigns a unique session identifier that is referred to as a ***session reference number*** to each session. ASP is an *asymmetrical protocol* that provides one set of services to the workstation and a different set of services to the server. The ASP workstation application always initiates the process of setting up a session and the communication across a session, and the ASP server replies to commands that it receives. (ASP is built on top of ATP, and it follows the transaction model of ATP while adding session-connection services.) The only case in which an ASP server can initiate communication is through the ASP attention mechanism. Figure 8-1 shows ASP and its underlying protocols.

Figure 8-1 ASP and its underlying protocols



Note
To open a session with an ASP server, you must know the server's internet socket address; you can use the Name-Binding Protocol (NBP) to obtain the internet socket address of any ASP server that advertises its services on the network. ◆

You can open an ASP session and send commands to the ASP server for a higher-level protocol, such as AFP, to interpret and execute. The commands that you send to an ASP server must follow the syntax prescribed by the higher-level protocol that is a client of the ASP server. ASP simply transfers the commands, and the ASP server returns a response.

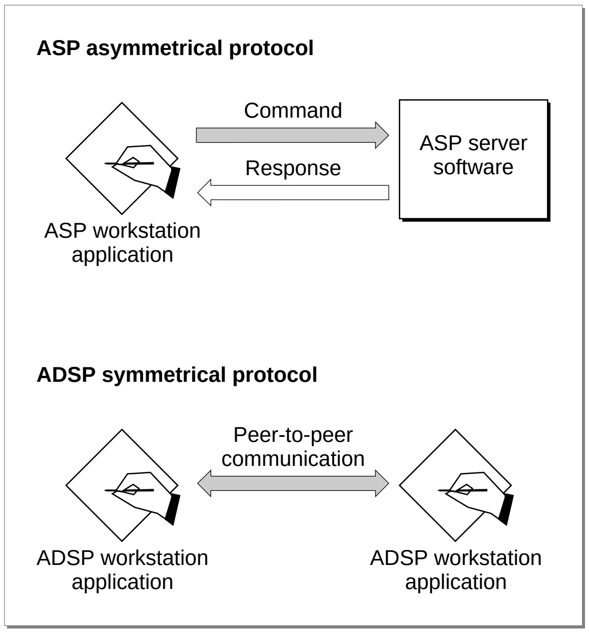
For example, the AppleShare server is AppleTalk's ASP server implementation. AFP uses the services of ASP to allow a user to manipulate files on an AppleShare server. (AFP is an example of an ASP workstation application.) As long as the ASP session is open, the workstation can send AFP commands to request directory information, change filenames, and so forth.

ASP ensures that commands from a workstation are delivered to the ASP server without duplication in the same order in which they were sent. This feature is useful for implementing applications that are *state dependent*, that is, applications in which the response to a request is dependent on a previous request. A workstation application connecting to a file server to read a file is an example of a state-dependent application: before the application can read the file, it must have first issued a request to open the file.

ASP also provides an attention mechanism that allows the server to send a message to the workstation. For example, a file server can use this messaging system to notify all of the workstations that are using the file server that it is shutting down. ASP is responsible for closing down the session if one end fails or becomes unreachable, and it will inform the workstation applications of its action. The .XPP driver implements ASP.

Once again, if your application requires a session-oriented protocol, you should consider whether to use ADSP instead of ASP. ASP and ADSP have in common the salient feature that they are both session-oriented protocols. However, they each provide a different type of session-oriented service. Although the differences between them are not parallel, in contrasting the two protocols it is helpful to recognize that ASP is limited by the structure of a transaction because it is built on top of ATP and that ADSP entails more flexibility because it is built directly on top of DDP. Figure 8-2 illustrates the different behavior and functions of the two protocols.

Figure 8-2 Differences between ASP and ADSP


ASP asymmetrical protocol
Command
Response
ASP server software
ASP workstation application
ADSP symmetrical protocol
Peer-to-peer communication
ADSP workstation application
ADSP workstation application

Please read this note before you continue

ASP provides an application programming interface for the workstation side only. The primary use of ASP is to provide services for the AppleTalk Filing Protocol (AFP). In most cases, you will not need to use ASP. Because very few application program developers use ASP, this chapter does not include a “Using” section. It includes only an overview of ASP and an ASP reference section. ◆

ASP Reference

This section describes the data structures and routines that are specific to the AppleTalk Session Protocol (ASP).

The “Data Structures” section shows the Pascal data structure for the XPP parameter block for ASP. The “Routines” section describes the routines for opening an ASP session, closing a specific ASP session or all ASP sessions on your node, sending commands and data across a session to a server, obtaining information about ASP sessions on your node or about a server, and canceling a request to open a session.

Data Structures

This section describes the XPP parameter block that ASP functions use to pass information to and receive it from the .XPP driver.

XPP Parameter Block for ASP

The ASP functions use the XPP parameter block defined by the `XPPParamBlock` data type to pass input and receive output parameters. In addition to the standard XPP parameter block fields, the ASP functions use variant records. The `ASPOpenSession` function uses the `ASPOpenPrm` variant record. The `ASPAbortOS` function uses the `ASPAbortPrm` variant record. The `ASPGetParms` function uses the `ASPSizeBlk` variant record. The `ASPUserCommand` and `ASPUserWrite` functions uses the `ASPSubPrm` variant record. The `ASPUserWrite`, `ASPUserCommand`, and `ASPGetStatus` functions use the `ASPEndPrm` variant record.

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

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

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

Field descriptions

`ioCompletion`	A pointer to a completion routine that you can provide. When you execute a function asynchronously, AppleTalk calls your completion routine when it completes execution of the function if you specify a pointer to the routine as the value of this field. Specify `NIL` for this field if you do not wish to provide a completion routine. If you execute a function synchronously, AppleTalk ignores the `ioCompletion` field. For information about completion routines, see the chapter "Introduction to AppleTalk" in this book.
`ioResult`	The result of the function. When you execute the function asynchronously, the function sets this field to 1 and returns a function result of `noErr` as soon as the function begins execution. When the function completes execution, it sets the `ioResult` field to the actual result code.
`ioRefNum`	The driver reference number for the .XPP driver. The Device Manager's `OpenDriver` function that you use to open the .XPP driver returns the driver reference number in the `refnum` field. You must supply this value. You can call this function to obtain the .XPP driver's reference number even if the .XPP driver is already open. The MPW interface does not fill in this value. For information on opening the .XPP driver, see the chapter "AppleTalk Utilities" in this book. For information on the `OpenDriver` function, see the chapter "Device Manager" in *Inside Macintosh: Devices.*
`csCode`	The command code of the XPP command to be executed. The MPW interface fills in this field.

Routines

This section describes the ASP functions that you use to

- open an ASP session from an ASP workstation application or process
- close one or all ASP sessions for a workstation from your ASP workstation application or process
- send commands and data across the session from the workstation to the server
- obtain information about the maximum capacities of the ASP implementation on your node, such as the number of concurrent ASP sessions and the amount of data that you can send
- obtain status information about a server without establishing a session with that server

Before you can open an ASP session or call any of the ASP functions, you must open the .XPP driver. You use the Device Manager's `OpenDriver` function to open the .XPP driver. The .MPP and .ATP drivers must be open before you open the .XPP driver. For information on opening the .XPP driver, see the chapter "AppleTalk Utilities" in this book. For information on the `OpenDriver` function, see the chapter "Device Manager" in *Inside Macintosh: Devices.*

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

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

An arrow preceding a parameter indicates whether the parameter is an input parameter, an output parameter, or both:

Arrow	**Meaning**
→	Input
←	Output
↔	Both

Opening and Closing ASP Sessions

This section describes how to open and close an ASP session from your workstation application or process. It includes

- the `ASPOpenSession` function that you use to open a session with a server
- the `ASPCloseSession` function that you use to close a single session when you are finished using the connection
- the `ASPCloseAll` function that you use to close all of the ASP sessions running on your node

ASPOpenSession

The `ASPOpenSession` function opens an ASP session between an ASP workstation application and an ASP server application.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `openSess` for this function.
←	`sessRefnum`	`Integer`	The session reference number.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`aspRetry`	`Byte`	The number of retries.
→	`serverAddr`	`AddrBlock`	The server socket address.
→	`scbPointer`	`Ptr`	A pointer to the session control block.
→	`attnRoutine`	`Ptr`	A pointer to an attention routine.

Field descriptions

`sessRefnum`	A unique number that the .XPP driver assigns to the session that it opens if the function completes successfully.
`aspTimeout`	The interval in seconds between retries of the open session request.
`aspRetry`	The number of times that ASP will retry to open a session.
`serverAddr`	The internet socket address of the socket that the server is using to listen for requests to open a session.
`scbPointer`	A pointer to a session control block (SCB) that the .XPP driver requires to maintain an open session. The `scbMemSize` constant defines the size of the session control block. The memory that you allocate for the SCB must be nonrelocatable or locked because it belongs to the .XPP driver for the life of the session.
`attnRoutine`	A pointer to a routine that ASP calls if the workstation component of ASP receives an attention request from the server or if the session is closed. If you do not want to specify an attention routine to be called, set this pointer to `NIL`.

DESCRIPTION

To gain access to an ASP server, you must call the `ASPOpenSession` function to open a session. Before calling the `ASPOpenSession` function, you must obtain the internet socket address of the socket that the ASP server uses to listen for incoming session requests. The server uses a session listening socket (SLS) for this purpose. You can use the Name-Binding Protocol (NBP) to get the internet socket address of an SLS. You pass the internet socket address of the SLS as the value of the `serverAddr` parameter.

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

- You have called the `ASPCloseSession` function to close the session and it has completed successfully.

- The server end of the ASP session has closed the session or the .XPP driver has closed the session. In both cases, the .XPP driver returns an `aspParamErr` result code as the result of a call for that session.

You can also pass the `ASPOpenSession` function a pointer to an attention routine that the .XPP driver calls when it receives an attention request from the server and when the session is closing. ASP provides an attention mechanism that allows the ASP server to notify the ASP workstation application of some event or critical piece of information. As the value of the `attnRoutine` parameter, you can specify a pointer to your attention routine, and the .XPP driver will call this routine when it receives an attention request from the server or when the server, the workstation, or ASP closes the session; ASP, as implemented in the .XPP driver, will close a session if it cannot successfully open the session before it exhausts the number of retries.

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

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

The .XPP driver calls your attention routine with

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

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

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

When the .XPP driver receives an `ASPOpenSession` function, it sends a special open session (`OpenSession`) packet as an ATP request to the SLS; this packet carries the address of the socket that the ASP workstation application or process is using for the session. The open session packet also carries a version number so that both ends can verify that they are using the same version of ASP.

Once you open a session, you can send commands and data to the server and receive command replies from the server. However, before you open an ASP session, you should call the `ASPGetParms` function to determine the maximum sizes of commands and replies that ASP supports on your node.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPOpenSession` function from assembly language, call the `_Control` trap macro with a value of `openSess` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

`aspBadVersNum`	–1066	The server cannot support the ASP version number
`aspNoMoreSess`	–1068	The .XPP driver cannot support another ASP session (the number of sessions that the driver is capable of supporting is dependent on the machine type)
`aspNoServers`	–1069	There is no server at the specified `serverAddr` address, or the server did not respond to the request
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspServerBusy`	–1071	The server cannot open another session
`reqAborted`	–1105	The `ASPOpenSession` function call was aborted by an `ASPAbortOS` function call

SEE ALSO

For information on how to use NBP, see the chapter "Name-Binding Protocol (NBP)" in this book.

You can use the `ASPAbortOS` function described on page 8-25 to cancel an outstanding `ASPOpenSession` function request before it completes execution.

For the maximum sizes of commands and replies that ASP supports on your node, use the `ASPGetParms` function, described on page 8-22.

ASPCloseSession

The `ASPCloseSession` function closes the session that you identify.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `closeSess` for this function.
→	`sessRefnum`	`Integer`	The session reference number.

Field descriptions

`sessRefnum`	A unique number that the .XPP driver assigned to this session when you called the `ASPOpenSession` function to open the session.

DESCRIPTION

To close a single session, you pass the session's reference number to the `ASPCloseSession` function in the `sessRefnum` field. The session reference number is the number that the .XPP driver assigns to the session and returns to you in the `sessRefnum` field when you open a session using the `ASPOpenSession` function. The `ASPCloseSession` function cancels any function calls that are pending for the session, closes the session, and calls the attention routine for the session, if there is one, with an attention code of 0 to indicate that the session is closed.

Note that there are other ways in which a session can be closed: for example, ASP closes a session when one end of the session fails. A session remains open until it is explicitly terminated by either the ASP workstation application or the ASP server or until one of the session's ends fails or becomes unreachable.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPCloseSession` function from assembly language, call the `_Control` trap macro with a value of `closeSess` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspSessClosed`	–1072	The .XPP driver is in the process of closing down the session

SEE ALSO

You can call the `ASPCloseAll` function, described next, to cancel all active ASP sessions on your node. Note that you should use the `ASPCloseAll` function cautiously as applications and processes other than your own that are running on the same node could be using ASP sessions.

ASPCloseAll

The `ASPCloseAll` function closes all of the active ASP sessions on the node.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `closeAll` for this function.

DESCRIPTION

To close all of the ASP sessions that are active and maintained by the .XPP driver on the node, you call the `ASPCloseAll` function. This function cancels all active requests, and it invokes the attention routines for any active sessions, if attention routines were provided. A good use of this function is as a system-level function call to ensure that all ASP sessions are closed before you close the .XPP driver.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPCloseAll` function from assembly language, call the `_Control` trap macro with a value of `closeAll` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspSessClosed`	–1072	The .XPP driver is in the process of closing down the session

Sending Commands and Writing Data From the Workstation to the Server

After you open a session, you can send a sequence of commands or a variable-size block of data across the session to the server. ASP returns to your ASP workstation application replies to the commands from the server end of the session. This section describes the `ASPUserCommand` function that you use to send commands to the server and the `ASPUserWrite` function that you use to send data.

ASPUserCommand

The `ASPUserCommand` function sends a command that you define from the workstation to the server across a session between them. ASP does not interpret the command syntax or execute the command; it simply transfers the command to the ASP server.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
←	`cmdResult`	`LongInt`	The ASP command result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `userCommand` for this function.
→	`sessRefnum`	`Integer`	The session reference number.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`cbSize`	`Integer`	The command block size.
→	`cbPtr`	`Ptr`	A pointer to the command block.
↔	`rbSize`	`Integer`	The reply buffer and reply size.
→	`rbPointer`	`Ptr`	A pointer to the reply buffer.
←	`ccbStart`	`Array`	The beginning of memory for the CCB.

Field descriptions

`cmdResult`	The ASP command result, consisting of 4 bytes of data returned by the server. The ASP client application defines the contents of the command result field. For example, AFP defines this field to specify the result of the AFP command. This field is valid if no system-level error is returned in the `ioResult` field.
`sessRefnum`	The reference number assigned to this session that the `ASPOpenSession` function returned when you called it to open the session.
`aspTimeout`	The time in seconds after which ASP is to retry to send the command across the session. You cannot specify the number of retries, just the time between them. ASP will retry to transmit the command until either it succeeds or the session is closed.
`cbSize`	The size in bytes of the buffer that contains the command that ASP is to send to the sever. The command buffer size must not exceed the value of `aspMaxCmdSize`, which the `ASPGetParms` function returns.
`cbPtr`	A pointer to a buffer containing the command that ASP is to send to the server.
`rbSize`	On input, the size in bytes of the buffer that you allocated to contain the command reply that you expect to receive from the server. On return, the size in bytes of the reply data that was actually returned.
`rbPointer`	A pointer to the buffer for the command reply.
`ccbStart`	The beginning of the memory for the command control block (CCB) that the .XPP driver is to use. The memory allocated for the CCB must not exceed the maximum of 150 bytes for this function. The CCB is an array that is part of the .XPP parameter block.

DESCRIPTION

You use the `ASPUserCommand` function to send a user command across an ASP session. You pass to the `ASPUserCommand` function a pointer to a variable-size command block that contains the command data to be sent to the ASP server. The command data must adhere to a format defined by a higher-level protocol that is built on top of the ASP server, such as the AppleTalk Filing Protocol (AFP). The command data requests the server to perform a particular function and return a reply consisting of a variable-size block of data and a command result. Some examples of the types of commands that you can send are

- a request to open a particular file on a file server (The server would return a small amount of data for this request.)
- a request to read a range of bytes from a device (The server might send a multiple-packet reply to this request.)

ASP delivers the commands in the same sequence that you send them. ASP does not interpret the command data or in any way participate in executing the command's function. It simply conveys the command data, included in a higher-level format, to the server end of the session and returns the command reply to your ASP workstation application. The command reply consists of a 4-byte command result returned in the `cmdResult` field and a variable-size command reply returned in the reply buffer that you supply. The higher-level protocol that is the client of ASP defines the content and use of the command result. A command result error is returned in the `cmdResult` field. All other types of errors are returned in the function's parameter block `ioResult` field. These error codes report the following error conditions:

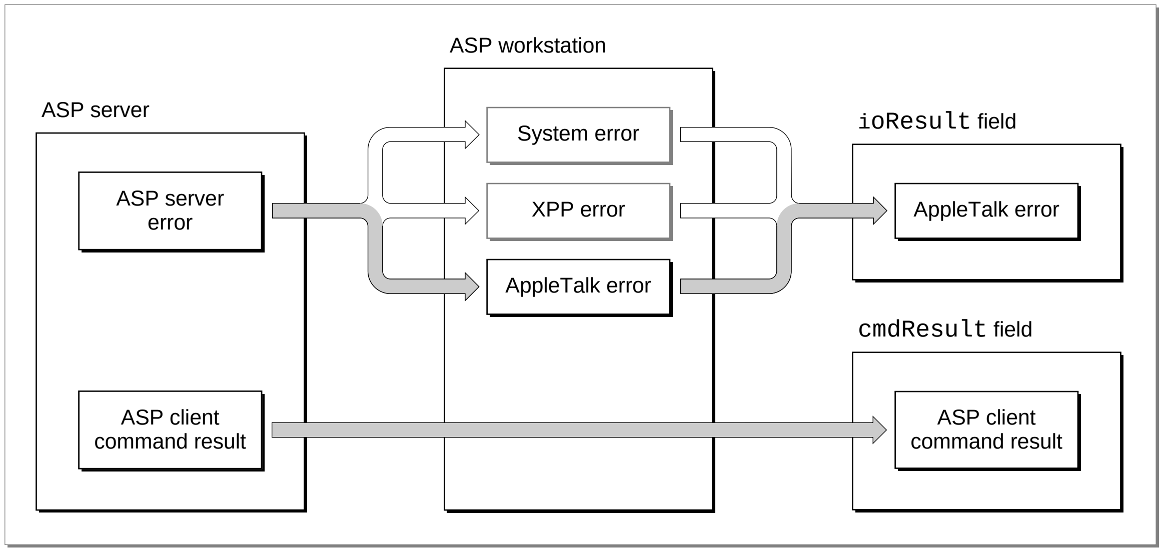
- system-level errors returned by the .XPP driver indicating, for example, that the driver is not open or that a particular system call is not supported
- .XPP driver errors indicating, for example, that the session is not open
- AppleTalk errors returned from the underlying AppleTalk protocols
- an ASP-specific error returned from an ASP server, for example, in response to a failed `ASPOpenSession` function

Figure 8-3 on page 8-18 shows how these errors are reported.

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

You can minimize the amount of memory that is used for the CCB in the queue element. To do this, you should understand how ASP uses this memory. ASP uses the CCB to build data structures, including parameter blocks and buffer data structures (BDS), that it needs in order to make function calls to the .ATP driver. (See the chapter "AppleTalk Transaction Protocol [ATP]" in this book for information on ATP and buffer data structures.) The exact size of the memory that ASP needs for the CCB depends on the size of the replies that you expect from the server, and in the case of the `ASPUserWrite` function, the size of the data to be written. For the `ASPUserCommand`, `ASPUserWrite`, and `ASPGetStatus` functions, ASP must set up a BDS to hold the reply information. The number of entries in the BDS that ASP creates is equal to the size of the reply buffer divided by 578 (the maximum number of data bytes per ATP response packet), rounded up. A BDS cannot exceed eight elements. In addition to a BDS, ASP uses the CCB memory for the queue element to call the .ATP driver.

Figure 8-3 Error reporting in ASP



You can use the following equations to determine the minimum size of a CCB for a function that includes a reply buffer (`rbSize`):

```
bdsSize = MIN (((rbSize DIV 578) + 1),8) * bdsEntrySz
ccbSize = ioQElSize + 4 + bdsSize
```

For functions, such as `ASPUserWrite`, ASP must create an additional BDS and queue element to use in sending the write data to the server. You can use the following equations to determine the minimum size of a CCB for an `ASPUserWrite` function; these equations take into account the reply buffer (`rbSize`) and write data size (`wdSize`):

```
wrBDSSize = MIN (((wdSize DIV 578) + 1),8) * bdsEntrySz
wrCCBSz = (2 * ioQElSize) + 4 + bdsSize + wrBDSSize
```

Note that `bdsEntrySz` is equal to 12 and `ioQelSize` is equal to 50.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPUserCommand` function from assembly language, call the `_Control` trap macro with a value of `userCommand` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

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

ASPUserWrite

The `ASPUserWrite` function transfers data from the workstation to the server across a specific session.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
←	`cmdResult`	`LongInt`	The ASP command result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `userWrite` for this function.
→	`sessRefnum`	`Integer`	The session reference number.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`cbSize`	`Integer`	The command block size.
→	`cbPtr`	`Ptr`	A pointer to command blocks.
↔	`rbSize`	`Integer`	The reply buffer size and reply size.
→	`rbPointer`	`Ptr`	A pointer to the reply buffer.
↔	`wdSize`	`Integer`	The write data size.
→	`wdPtr`	`Ptr`	The write data pointer.
←	`ccbStart`	`Array`	The beginning of memory for the CCB.

Field descriptions

`cmdResult`	The ASP command result consisting of 4 bytes of data returned by the server. The ASP client application defines the contents of the command result field. For example, AFP defines this field to specify the result of the AFP command. This field is valid if no system-level error is returned in the `ioResult` field.

`sessRefnum`	The reference number of the session that you want to use to transfer data. The session reference number is the unique number that the .XPP driver assigned to this session when you opened the session by calling the `ASPOpenSession` function.
`aspTimeout`	The time in seconds after which ASP is to retry to send data across the session.
`cbSize`	The size in bytes of the command data that ASP is to transfer across the session.
`cbPtr`	A pointer to the buffer containing the command data to be transferred.
`rbSize`	On input, the size in bytes of the buffer that you allocated to contain the command reply that you expect to receive from the server. On return, the size in bytes of the reply data that was actually returned.
`rbPointer`	A pointer to the buffer for the reply data.
`wdSize`	On input, the size in bytes of the of the write data that the command is to send. On return, the size in bytes of the write data that was actually sent.
`wdPtr`	A pointer to the buffer containing the data to be written.
`ccbStart`	The beginning of the memory for the command control block (CCB) that the .XPP driver is to use. The maximum size of this block is 296 bytes. The CCB is an array that is part of the .XPP parameter block.

DESCRIPTION

The `ASPUserWrite` function allows you to transfer a variable-size block of data to the server end of the ASP session and receive a reply. If you have previously called the `ASPUserCommand` function to send a command that directs the ASP server to open a file, you can call the `ASPUserWrite` function to write data to the file.

The .XPP driver uses the memory at the end of the XPP parameter block defined as a `CCBStart` array as an internal command control block (CCB). To ensure that the function executes successfully, you can specify the maximum size for this array as indicated in particular for the function that uses it. If you want to limit the amount of memory used for the CCB, you can specify the minimum amount of memory required for this array.

A command result error is returned in the `cmdResult` field. All other types of errors are returned in the function's parameter block `ioResult` field. Error reporting for the `ASPUserWrite` function is the same as for the `ASPUserCommand`.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPUserWrite` function from assembly language, call the `_Control` trap macro with a value of `userWrite` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

`aspBufTooSmall`	–1067	The reply data exceeds the size of the reply buffer; the .XPP driver will fill the buffer and truncate the data
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspSessClosed`	–1072	The .XPP driver is in the process of closing the session
`aspSizeErr`	–1073	The size of the command block exceeds the maximum size of 296 bytes

SEE ALSO

To send a command to the server to direct it to perform a prerequisite action before you use the `ASPUserWrite` command to write data, use the `ASPUserCommand` function, described on page 8-15. To determine the minimum amount of memory required for the CCB or to find out more about the possible types of error conditions for which errors are returned and how these error results are reported, see the description of the `ASPUserCommand` function.

Obtaining Information About ASP's Maximum Capacities and the Status of the Server

This section describes the `ASPGetParms` function that you can use to determine how many concurrent ASP sessions can run on your node and the maximum amount of data that you can send and receive across a session. Before you open an ASP session, you should call the `ASPGetParms` function to determine the maximum sizes of commands and replies that ASP supports on your node.

This section also describes the `ASPGetStatus` function that you can use to obtain server status information without opening a session with the server.

ASPGetParms

The `ASPGetParms` function returns the maximum size of the data that you can send and receive across an ASP session and the maximum number of concurrent ASP sessions that the .XPP driver running on your node supports.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `getParms` for this function.
←	`aspMaxCmdSize`	`Integer`	The maximum size of command data.
←	`aspQuantumSize`	`Integer`	The maximum data size.
←	`numSesss`	`Integer`	The number of sessions.

Field descriptions

`aspMaxCmdSize`	The maximum size in bytes of a command that you can send to the server.
`aspQuantumSize`	The maximum size in bytes of the data that you can either request ASP to transfer to the server in an `ASPUserWrite` function call or receive from the server in a command reply.
`numSesss`	The number of concurrent ASP sessions that the .XPP driver supports on your node.

DESCRIPTION

The `ASPGetParms` function returns information about the data capacity of an ASP session that you need to know to send commands using the `ASPUserCommand` and `ASPUserWrite` functions and write data using the `ASPUserWrite` function. It also tells you how many concurrent ASP sessions your node supports. You do not need to establish a session before you call the `ASPGetParms` function.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPGetParms` function from assembly language, call the `_Control` trap macro with a value of `getParms` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

`noErr`	0	No error

ASPGetStatus

The `ASPGetStatus` function returns status information about the server whose internet socket address you provide.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `getStatus` for this function.
→	`aspTimeout`	`Byte`	The retry interval in seconds.
→	`aspRetry`	`Byte`	The number of retries.
→	`serverAddr`	`AddrBlock`	The server socket address.
↔	`rbSize`	`Integer`	The reply buffer and reply size.
→	`rbPtr`	`Ptr`	A pointer to the reply buffer.
←	`ccbStart`	`Array`	The beginning of memory for the CCB.

Field descriptions

`aspTimeout`	The time in seconds after which ASP is to retry to obtain information about the status of the server whose address you provide.
`aspRetry`	The number of times ASP is to retry to obtain the server status information.
`serverAddr`	The internet socket address of the server about which you want status information.
`rbSize`	On input, the size in bytes of the buffer that you allocated to contain the reply that you expect to receive from the server. On return, the size in bytes of the reply (status) data that was actually returned.

`rbPtr`	A pointer to the buffer for the reply data.
`ccbStart`	The beginning of the memory for the command control block (CCB) that the .XPP driver is to use. The memory allocated for the CCB must not exceed the maximum of 150 bytes.

DESCRIPTION

You can use the `ASPGetStatus` function to obtain service status information about a server without opening a session between your application and that server. ASP does not impose any structure on the status block. The protocol above ASP defines the structure. The .XPP driver uses the memory at the end of the XPP parameter block defined as a `CCBStart` array as an internal command control block (CCB). To ensure that the function executes successfully, you can specify the maximum size for this array as indicated in particular for the function that uses it. If you want to limit the amount of memory used for the CCB, you can specify the minimum amount of memory required for this array.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPGetStatus` function from assembly language, call the `_Control` trap macro with a value of `getStatus` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

`aspBufTooSmall`	–1067	The reply data exceeds the size of the reply buffer; the .XPP driver will fill the buffer and truncate the data
`aspNoServer`	–1069	There was no response from the server whose address you specified as the value of `serverAddr`

SEE ALSO

To determine the minimum amount of memory required for the CCB, refer to the description of the `ASPUserCommand` function on page 8-15.

Canceling an ASP Request to Open a Session

This section describes the `ASPAbortOS` function that you can use to cancel a pending request to open a session.

ASPAbortOS

The `ASPAbortOS` function cancels a specific pending request to open an ASP session function.

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

`thePBptr`	A pointer to an XPP parameter block.
`async`	A Boolean that specifies whether the function should be executed asynchronously or synchronously. Specify `TRUE` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	A pointer to a completion routine.
←	`ioResult`	`OSErr`	The function result.
→	`ioRefNum`	`Integer`	The .XPP driver reference number.
→	`csCode`	`Integer`	Always `abortOS` for this function.
→	`abortSCBPointer`	`Ptr`	A pointer to the session control block.

Field descriptions

`abortSCBPointer`

A pointer to the session control block (SCB) that you passed to the `ASPOpenSession` function that you want to cancel.

DESCRIPTION

The `ASPAbortOS` function cancels a single call to the `ASPOpenSession` function if that function has not yet completed execution. You identify the request to be canceled by passing the `ASPAbortOS` function the pointer to the original session control block that you specified to open the session.

SPECIAL CONSIDERATIONS

Note that you must provide the .XPP driver reference number as an input parameter to this function. You can obtain the driver reference number by calling the Device Manager's `OpenDriver` function.

ASSEMBLY-LANGUAGE INFORMATION

To execute the `ASPAbortOS` function from assembly language, call the `_Control` trap macro with a value of `abortOS` in the `csCode` field of the parameter block. You must also specify the .XPP driver reference number. To execute the `_Control` trap asynchronously, include the value `,ASYNC` in the operand field.

RESULT CODES

`cbNotFound`	–1102	Specified SCB was not found (there is no outstanding open session function call with this SCB)

SEE ALSO

For information on the session control block, see the description of the `ASPOpenSession` function on page 8-9.

Summary of ASP

Pascal Summary

Constants

```
CONST
   {.XPP driver unit and reference number}
   xppUnitNum     =      40;           {XPP unit number}
   xppRefNum      =     -41;           {XPP reference number}

   {command codes for ASP}
   openSess       =     255;           {open session}
   closeSess      =     254;           {close session}
   userCommand    =     253;           {user command}
   userWrite      =     252;           {user write}
   getStatus      =     251;           {get server status}
   getParms       =     249;           {get parameters for session}
   abortOS        =     248;           {cancel open session request}
   closeAll       =     247;           {close all open sessions}

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

Data Types

Address Block Record

```
TYPE AddrBlock =
   PACKED RECORD
      aNet:       Integer;          {network number}
      aNode:      Byte;             {node ID}
      aSocket:    Byte;             {socket number}
   END;
```

XPP Parameter Block for ASP

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

TYPE XPPParamBlock =
   PACKED RECORD
      qLink:            QElemPtr;         {reserved}
      qType:            Integer;          {reserved}
      ioTrap:           Integer;          {reserved}
      ioCmdAddr:        Ptr;              {reserved}
      ioCompletion:     ProcPtr;          {completion routine}
      ioResult:         OSErr;            {result code}
      cmdResult:        LongInt;          {command result (ATP user bytes)}
      ioVRefNum:        Integer;          {reserved}
      ioRefNum:         Integer;          {driver reference number}
      csCode:           Integer;          {call command code}
      CASE XPPPrmBlkType OF
      ASPAbortPrm:
         (abortSCBPtr: Ptr);              {SCB pointer for AbortOS}
      ASPSizeBlk:
         (aspMaxCmdSize: Integer;         {maximum size of data for commands}
         aspQuantumSize:Integer;          {maximum size of data for request }
                                          { commands and receive replies}
         numSesss: Integer);              {number of concurrent sessions }
                                          { for your node}
   }
      XPPPrmBlk:
      (sessRefnum: Integer;               {offset to session refnum}
      aspTimeout: Byte;                   {timeout for ATP}
      aspRetry: Byte;                     {retry count for ATP}
      CASE XPPSubPrmType OF
      ASPOpenPrm:
      (serverAddr: AddrBlock;             {server address block}
      scbPointer: Ptr;                    {SCB pointer}
      attnRoutine: Ptr);                  {attention routine pointer}
      ASPSubPrm:
      (cbSize: Integer;                   {command block size}
      cbPtr: Ptr;                         {command block pointer}
      rbSize: Integer;                    {reply buffer size}
      rbPtr: Ptr;                         {reply buffer pointer}
      CASE XPPEndPrmType OF
      ASPEndPrm:
      (wdSize: Integer;                  {write data size}
      wdPtr: Ptr;                        {write data pointer}
      ccbStart: ARRAY[0..295] OF Byte))); {beginning of command control }
                                         { block}
   END;

XPPParmBlkPtr = ^XPPParamBlock;
```

Routines

Opening and Closing ASP Sessions

```
FUNCTION ASPOpenSession     (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
FUNCTION ASPCloseSession    (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
FUNCTION ASPCloseAll        (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
```

Sending Commands and Writing Data From the Workstation to the Server

```
FUNCTION ASPUserCommand     (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
FUNCTION ASPUserWrite       (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
```

Obtaining Information About ASP's Maximum Capacities and the Status of the Server

```
FUNCTION ASPGetParms        (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
FUNCTION ASPGetStatus       (thePBptr: XPPParmBlkPtr; async: Boolean): OSErr;
```

Canceling an ASP Request to Open a Session

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

C Summary

Constants

```
enum {                                   /*.XPP driver unit and reference */
                                         /* number*/
   xppUnitNum     =         40,          /*XPP unit number*/
   xppRefNum      =        -41};         /*XPP reference number*/
```

```
enum {                                    /*command codes for ASP*/
   openSess    =        255,              /*open session*/
   closeSess   =        254,              /*close session*/
   userCommand =        253,              /*user command*/
   userWrite   =        252,              /*user write*/
   getStatus   =        251,              /*get status*/
   getParms    =        249,              /*get parameters*/
   abortOS     =        248,              /*cancel open session request*/
   closeAll    =        247};             /*close all open sessions*/

enum {                                    /*miscellaneous*/
xppLoadedBit   =        5,                /*XPP bit in PortBUse*/
scbMemSize     =        192};             /*size of memory for SCB*/
```

Data Types

Address Block Record

```
struct AddrBlock {
   short             aNet;                /*network name*/
   unsigned char     aNode;               /*node name*/
   unsigned char     aSocket;             /*socket number*/
};
```

XPP Parameter Block for ASP

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

typedef struct {
   XPPPBHeader
      short       sessRefnum;             /*offset to session refnum*/
      char        aspTimeout;             /*timeout for ATP*/
      char        aspRetry;               /*retry count for ATP*/
      short       cbSize;                 /*command block size*/
      Ptr         cbPtr;                  /*command block pointer*/
      short       rbSize;                 /*reply buffer size*/
      Ptr         rbPtr;                  /*reply buffer pointer*/
      short       wdSize;                 /*write data size*/
      Ptr         wdPtr;                  /*write data pointer*/
      char        ccbStart[296];          /*CCB memory allocated for */
                                          /* beginning of command control */
                                          /* block*/
}XPPPrmBlk;

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

typedef struct {
   XPPPBHeader
   Ptr            abortSCBPtr;            /*SCB pointer for ASPAbortOS*/
}ASPAbortPrm;

typedef struct {
   XPPPBHeader
   short          aspMaxCmdSize;       /*maximum size of data for commands*/
   short          aspQuantumSize;      /*maximum size of data for request */
                                       /* commands and receive replies*/
   short          numSesss;            /*number of concurrent sessions */
                                       /* for your node*/
}ASPGetparmsBlk;
```

Routines

Opening and Closing ASP Sessions

```
pascal OSErr ASPOpenSession  (ASPOpenPrmPtr thePBptr, Boolean async);
pascal OSErr ASPCloseSession (XPPParmBlkPtr thePBptr, Boolean async);
pascal OSErr ASPCloseAll     (XPPParmBlkPtr thePBptr, Boolean async);
```

Sending Commands and Writing Data From the Workstation to the Server

```
pascal OSErr ASPUserCommand (XPPParmBlkPtr thePBptr, Boolean async);
pascal OSErr ASPUserWrite   (XPPParmBlkPtr thePBptr, Boolean async);
```

Obtaining Information About ASP's Maximum Capacities and the Status of the Server

```
pascal OSErr ASPGetParms    (XPPParmBlkPtr thePBptr, Boolean async);
pascal OSErr ASPGetStatus   (XPPParmBlkPtr thePBptr, Boolean async);
```

Canceling an ASP Request to Open a Session

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

Assembly-Language Summary

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   2            ;size of data part
```

Command Codes (csCodes)

```
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
getParms          EQU   249            ;get parameters
abortOS           EQU   248            ;abort open session request
closeAll          EQU   247            ;close all open sessions
```

ASP Commands

```
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)
```

Miscellaneous

```
aspVersion        EQU   $0100                ;ASP version number
maxCmdSize        EQU   atpMaxData           ;maximum command block size
quantumSize       EQU   atpMaxData*atpMaxNum ;maximum reply size
tickleInt         EQU   30                   ;tickle interval (secs)
tickleTime        EQU   tickleInt*60*4       ;tickle timeout (ticks)
xppLoadedBit      EQU   atpLoadedBit+1       ;XPP loaded bit number in
                                             ; PortBUse
```

Data Structures

XPP Parameter Block Common Fields for ASP Routines

0	`qLink`	long	reserved
4	`qType`	word	reserved
6	`ioTrap`	word	reserved
8	`ioCmdAddr`	long	reserved
12	`ioCompletion`	long	address of completion routine
16	`ioResult`	word	result code
18	`cmdResult`	long	pointer to attention routine
22	`ioVRefNum`	word	reserved
24	`ioRefNum`	word	driver reference number

ASPOpenSession Parameter Block

26	`csCode`	word	command code; always `openSess`
28	`sessRefnum`	word	session reference number
30	`aspTimeout`	byte	retry interval in seconds
31	`aspRetry`	byte	number of retries
32	`serverAddr`	long	server internet socket address
36	`scbPointer`	pointer	pointer to session control block
40	`attnRoutine`	long	pointer to attention routine

ASPCloseSession Parameter Block

26	`csCode`	word	command code; always `closeSess`
28	`sessRefnum`	word	session reference number

ASPCloseAll Parameter Block

26	`csCode`	word	command code; always `closeAll`

ASPUserCommand Parameter Block

18	`cmdResult`	long	ASP command result
26	`csCode`	word	command code; always `userCommand`
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 and reply size
40	`rbPtr`	pointer	pointer to reply buffer
50	`ccbStart`	record	start of memory for CCB

ASPUserWrite Parameter Block

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

ASPGetParms Parameter Block

26	`csCode`	word	command code; always `getParms`
28	`aspMaxCmdSize`	word	maximum size of command block
30	`aspQuantumSize`	word	maximum data size
32	`numSesss`	word	maximum number of sessions

ASPGetStatus Parameter Block

26	`csCode`	word	command code; always `getStatus`
30	`aspTimeout`	byte	retry interval in seconds
31	`aspRetry`	byte	number of retries
32	`serverAddr`	long	server internet socket address
38	`rbSize`	word	reply buffer and reply size
40	`rbPtr`	pointer	pointer to reply buffer
50	`ccbStart`	record	start of memory for CCB

ASPAbortOS Parameter Block

26	`csCode`	word	command code; always `abortOS`
28	`abortSCBPtr`	pointer	pointer to session control block

Result Codes

`noErr`	0	No error
`aspBadVersNum`	–1066	The server cannot support the ASP version number
`aspBufTooSmall`	–1067	The reply data exceeds the size of the reply buffer; the .XPP driver will fill the buffer and truncate the data
`aspNoMoreSess`	–1068	The .XPP driver cannot support another ASP session (the number of sessions that the driver is capable of supporting is dependent on the machine type)
`aspNoServers`	–1069	There is no server at the specified `serverAddr` address, or the server did not respond to the request
`aspParamErr`	–1070	You specified an invalid session reference number, or the session has been closed
`aspServerBusy`	–1071	The server cannot open another session
`aspSessClosed`	–1072	The .XPP driver is in the process of closing down the session
`aspSizeErr`	–1073	The size of the command block exceeds the maximum size of `aspMaxCmdSize`
`cbNotFound`	–1102	Specified SCB was not found (there is no outstanding open session function call with this SCB)
`reqAborted`	–1105	The `ASPOpenSession` function call was aborted by an `ASPAbortOS` function call