Using The AppleTalk Manager Implementing AppleTalk

This section describes how to determine whether AppleTalk Phase 2 drivers are present and gives some advice on how to select the AppleTalk protocol that best serves your purposes.

Finding Out if AppleTalk Phase 2 Drivers Are Present

Once **The .MPP Driver** has been loaded into memory, you can use the **Gestalt** function with the gestaltAppleTalkVersion selector to check the version of AppleTalk. The **Gestalt** function returns the current version of **The .MPP Driver**. If the version is equal to or greater than the number 53, then **The .MPP Driver** supports AppleTalk Phase 2, and you can assume the other Phase 2 drivers are present.

Alternatively, you can call the **SysEnvirons** function. If the atDrvrVersNum field of the **SysEnvRec** data structure returned by this function is equal to or greater than 53, then **The .MPP Driver** supports AppleTalk Phase 2.

The ExtendedBit flag returned by the **PGetAppleTalkInfo** function is TRUE if the node is connected to an extended AppleTalk network. (The ExtendedBit flag is bit 15 of the configuration parameter returned by this function.) Note that the presence of the AppleTalk Phase 2 drivers does not of itself indicate that the node is connected to an extended network.

Deciding Which AppleTalk Protocol to Use

AppleTalk offers a variety of communications protocols at a variety of levels. Your choice of protocol or protocols to use depends primarily on your needs and can be influenced by your familiarity with network communications in general.

You can write your own protocol handlers and call the low-level AppleTalk device drivers directly. However, if you are not a communications expert and have no desire to design your own network protocols, you should probably use one of three AppleTalk protocols for sending and receiving data over the AppleTalk internet: the AppleTalk Transaction Protocol (ATP), the AppleTalk Session Protocol (ASP), or **AppleTalk Data Stream Protocol (ADSP)**.

ATP is a lower-level protocol than ASP or **ADSP**. You cannot use ATP to establish a session and keep it open; rather, you request data from another socket client or send a response (up to eight packets of data) from your socket to another socket client that has requested data. You should use ATP if you want only to send a small amount of data and do not need the overhead required to maintain an open connection.

ASP is designed to support a session between a server and one or more workstations. It is an asymmetrical protocol: all exchanges are initiated by a workstation and responded to by a server. The server cannot initiate an exchange of data except to send to a workstation an attention message that directs the workstation to request data from the server. An application running on a workstation must make calls to ASP to communicate with any server that uses ASP. If you want to develop a new type of asymmetrical, transaction-oriented server, you should consider using ASP to implement it.

ADSP is a symmetrical protocol that you can use to establish and maintain a connection between two equal entities (a peer-to-peer connection). Either end of an **ADSP** connection can send data at any time. Although **ADSP** is a client of DDP and therefore sends and receives data in packets (as do ATP and ASP), to an application using **ADSP** the data appears to be sent and received as a continuous stream. In addition to the duplex data stream maintained by an **ADSP** session, **ADSP** allows either end of a connection to send an attention message to the other end. You can use **ADSP** to establish two-way communication between computers, such as an interoffice party line or a terminal emulation program. If you want to develop an application that requires two-way communication, you should consider using **ADSP** to implement it.