

Windows Sockets

Windows Sockets for Appletalk

**An Open Interface for
Appletalk Network Programming under
Microsoft Windows NT**

Version 1.2

1 December 1992

**Authors: Nikhil Kamkolkar (nikhilk@microsoft.com)
Jameel Hyder (jameelh@microsoft.com)**

Copyright 1992 by Microsoft Corporation
All rights reserved.

This document may be freely redistributed in any form, electronic or otherwise, provided that it is distributed in its entirety and that the copyright and this notice are included.

Revision history:

1.2 20 Jul 1994 JameelHyder
(jameelh@microsoft.com)
Review comments, errors and omissions.

Windows Sockets for AppleTalk

Version 1.2

1. INTRODUCTION	1
1.1 What is Windows Sockets for AppleTalk?	1
1.2 Revision History	1
2. PROGRAMMING WITH SOCKETS FOR APPLETALK	2
2.1 Sockets (SOCKADDR_AT structure)	2
2.1.1 SOCK_STREAM	2
2.1.2 SOCK_RDM	2
2.1.3 SOCK_DGRAM	2
2.2 Out-of-band data	3
2.3 Broadcasting	3
2.3 Byte Ordering	3
2.4 Socket Options	3
2.4.1 SO_REGISTER_NAME	4
2.4.2 SO_DEREGISTER_NAME	5
2.4.3 SO_LOOKUP_NAME	6
2.4.4 SO_CONFIRM_NAME	8
2.4.5 SO_LOOKUP_MYZONE	9
2.4.6 SO_LOOKUP_ZONES	10
2.4.7 SO_LOOKUP_ZONES_ON_ADAPTER (LOCAL ZONES)	11
2.4.8 SO_LOOKUP_NETDEF_ON_ADAPTER	12
2.4.9 SO_PAP_SET_SERVER_STATUS	12
2.4.10 SO_PAP_GET_SERVER_STATUS	13
2.4.11 SO_PAP_PRIME_READ	14
2.5 Database Files	15
2.6 Maximum number of sockets supported	15
3. SOCKET LIBRARY OVERVIEW	16
3.1 Socket Functions	16
4. SOCKET LIBRARY REFERENCE	17
4.1 Socket Routines	17

4.1.1 accept()	18	
4.1.2 bind()	19	
4.1.3 closesocket()	20	
4.1.4 connect()	21	
4.1.5 getpeername()	22	
4.1.6 getsockname()	23	
4.1.7 getsockopt()	24	
4.1.8 htonl()	25	
4.1.9 htons()	26	
4.1.10 inet_addr()		27
4.1.11 inet_ntoa()		28
4.1.12 ioctlsocket()	29	
4.1.13 listen()	30	
4.1.14 ntohl()	31	
4.1.15 ntohs()	32	
4.1.16 recv()/WsaRecvEx()	33	
4.1.17 recvfrom()		34
4.1.18 select()	35	
4.1.19 send()	36	
4.1.20 sendto()	37	
4.1.21 setsockopt()	38	
4.1.22 shutdown()	39	
4.1.23 socket()	40	
Appendix A. Error Codes and Header Files		41
A.1 Error Codes		41
A.2 Header Files		41
A.2.1 AppleTalk Header File	41	
Appendix B. Notes for Windows Sockets For AppleTalk Users		42
B.1 Introduction		42
B.2 Windows Sockets For AppleTalk Components		42

1. INTRODUCTION

1.1 What is Windows Sockets for AppleTalk?

This document provides necessary information to program over the AppleTalk protocols using Windows Sockets. In particular, the AppleTalk Data Stream Protocol (ADSP), the Datagram Delivery Protocol (DDP) and the Printer Access Protocol (PAP) are available through the Windows Sockets API. Necessary Name Binding Protocol (NBP) and Zone Information Protocol (ZIP) primitives are also available.

This document provides the information necessary to program to the Windows Sockets interface to use the above protocols, and it provides information about the necessary programming paradigm on the client side if the client happens to provide only the primitive protocols and does not provide a Windows Sockets library on top of them.

It is assumed that the reader will be familiar with the Windows Sockets 1.1 specification after which this document is structured.

1.2 Revision History

Incorporated review comments. Corrected errors in code samples. Added new aliases to various socket types.

2. PROGRAMMING WITH SOCKETS FOR APPLETALK

2.1 Sockets (SOCKADDR_AT structure)

The following section describes the mapping between a Windows Sockets idea of a socket and the AppleTalk sockets. The basic building block for communication in the Windows Sockets environment is the socket. Note that opening a Winsock socket will not actually create an AppleTalk socket. The ***bind*** operation creates the endpoint in the AppleTalk world. All AppleTalk-specific operations (socket options) need to happen after the bind operation.

AppleTalk uses the following socket address structure.

```
typedef struct sockaddr_at  
{  
    u_short      sat_family;  
    u_short      sat_net;  
    u_char       sat_node;  
    u_char       sat_socket;  
} SOCKADDR_AT, *PSOCKADDR_AT;
```

sat_family should be equal to PF_APPLETALK.

sat_net indicates the network number of the destination AppleTalk socket.

sat_node indicates the node number of the destination AppleTalk socket.

sat_socket indicates the socket number of the destination AppleTalk socket.

This socket can be of three types.

2.1.1 SOCK_STREAM

This socket type supports ADSP stream mode operation. Note that it is still the application's responsibility to specify the MSG_PARTIAL flag on every send on either side of the connection. The protocol number to be specified in the socket call is given below. Only one protocol number is valid for SOCK_STREAM in the AppleTalk protocol family.

ATPROTO_ADSP: Defined in the atalkwsh.h header file.

2.1.2 SOCK_RDM

This socket type supports ADSP message mode operation. The protocol numbers that can be specified in the socket call are given below.

ATPROTO_ADSP: Gives access to the ADSP protocol.

ATPROTO_PAP: Gives access to the PAP protocol.

2.1.3 SOCK_DGRAM

This socket type supports DDP operation. The protocol numbers that can be specified are defined in atalkwsh.h. Since DDP supports upto 255 protocol numbers, the range that is valid is given below. This protocol number will be the protocol number used in datagrams sent out on this socket, and only datagrams with this protocol number will be accepted on this socket. ATPROTO_BASE must be added to the desired protocol number.

Valid range: [ATPROTO_BASE + 1, ATPROTO_BASE+255]

2.2 Out-of-band data

ADSP supports transfer of expedited data. This is translated in the Windows Sockets world as Out-band-data. Both the SOCK_STREAM and SOCK_RDM will support receiving and sending out-of-band data. When a client receives indication of received out-of-band data, a receive must be posted for the maximum amount of expedited data possible in a single receive/send, i.e. the maximum size of the attention data plus the attention code (574 bytes). If the receive buffer size is less than the size of the received out-of-band data, the receive will be satisfied with as much of the received data as will fit in the buffer and the remaining data is dropped. It is possible to peek at out-of-band data.

2.3 Broadcasting

By using a datagram socket (SOCK_DGRAM), it is possible to send broadcast packets. To broadcast packets on an AppleTalk network, the destination node address must be 0xFF. The broadcast then happens on the network indicated by the network number.

2.4 Byte Ordering

All information coming in to the Windows Sockets API is assumed to be in host order. The only place where this becomes relevant is when storing/retrieving the ADSP attention code. This should be stored in the host order, and will be in the host order for incoming data.

2.5 Socket Options

Some AppleTalk primitives essential to its operation are provided in the form of socket options. The following extra options are supported for each socket type/protocol combination. These options are not necessarily valid for both a getsockopt and a setsockopt.

The following structures are used for the various options:

```
typedef union
{
    struct
    {
        USHORT    Network;
        UCHAR     Node;
        UCHAR     Socket;
    };
    ULONG        Address;
} WSH_ATALK_ADDRESS, *PWSH_ATALK_ADDRESS;
```

```
#define MAX_ENTITY 32
typedef struct {
    CHAR    ObjectNameLen;
    CHAR    ObjectName[MAX_ENTITY];
    CHAR    TypeNameLen;
    CHAR    TypeName[MAX_ENTITY];
    CHAR    ZoneNameLen;
    CHAR    ZoneName[MAX_ENTITY];
} WSH_NBP_NAME, *PWSH_NBP_NAME;
```

```
typedef struct
{
    WSH_ATALK_ADDRESS    Address;
    USHORT               Enumerator;
    WSH_NBP_NAME          NbpName;
} WSH_NBP_TUPLE, *PWSH_NBP_TUPLE;
```

```

typedef WSH_NBP_NAME  WSH_REGISTER_NAME, *PWSH_REGISTER_NAME;
typedef WSH_NBP_NAME  WSH_DEREGISTER_NAME, *PWSH_DEREGISTER_NAME;
typedef      WSH_NBP_NAME  WSH_REMOVE_NAME, *PWSH_REMOVE_NAME;
typedef struct _WSH_LOOKUP_ZONES
{
    ULONG  NoZones;

    // CHAR  Zones[] - null separated zones
} WSH_LOOKUP_ZONES, *PWSH_LOOKUP_ZONES;

typedef struct _WSH_LOOKUP_NETDEF_ON_ADAPTER
{
    USHORT  NetworkRangeLowerEnd;
    USHORT  NetworkRangeUpperEnd;

    // This will be followed by a null terminated ANSI default zone.
    // PCHAR  DefaultZone[]
} WSH_LOOKUP_NETDEF_ON_ADAPTER , *PWSH_LOOKUP_NETDEF_ON_ADAPTER;

typedef struct _WSH_LOOKUP_NAME
{
    WSH_NBP_TUPLE  LookupTuple;
    ULONG  NoTuples;

    // Array of NoTuple WSH_NBP_TUPLES
} WSH_LOOKUP_NAME, *PWSH_LOOKUP_NAME;

typedef struct _WSH_PAP_GET_SERVER_STATUS
{
    SOCKADDR_AT  ServerAddr;
    UCHAR        Reserved[PAP_UNUSED_STATUS_BYTES];
    UCHAR        ServerStatus[MAX_PAP_STATUS_SIZE+1];
} WSH_PAP_GET_SERVER_STATUS, *PWSH_PAP_GET_SERVER_STATUS;

```


2.5.1 SO_REGISTER_NAME

```
int PASCAL FAR setsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR * optval,
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be registered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_REGISTERNAME
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_REGISTER_NAME.
<i>optlen</i>	The size of the <i>optval</i> buffer (sizeof (WSH_REGISTER_NAME)).

Remarks This will register the name specified in the structure on the bound socket specified. If the name is currently registered on the net, this call will return an error.

Network Information

On a routed network upto ten NBP broadcast request packets are sent to a-router spaced one second apart unless the name collides with an existing name on the net. On a non-routed network lookup requests are sent instead of broadcast requests.

Error Codes	WSAEADDRINUSE	The name is already in use on the net.
--------------------	---------------	--

Sample Code

```
WSH_REGISTER_NAME  regName;

// SETSOCKOPT (register name)

regName.ObjectNameLen = strlen("Windows Socket Server");
regName.TypeNameLen = strlen("Windows Socket Type");
regName.ZoneNameLen = strlen("BLDG1 Zone");
strcpy(regName.ObjectName, "Windows Socket Server");
strcpy(regName.TypeName, "Windows Socket Type");
strcpy(regName.ZoneName, "BLDG1 Zone");

bytesWritten = sizeof(WSH_REGISTER_NAME);

err = setsockopt(listenHandle,
                SOL_APPLETALK,
                SO_REGISTER_NAME,
                (char *)&regName,
                bytesWritten );

if ( err == SOCKET_ERROR )
    printf( "failed: %ld\n", GetLastError( ));
```

2.5.2 SO_DEREGISTER_NAME, SO_REMOVE_NAME

SO_DEREGISTER_NAME and SO_REMOVE_NAME can be interchangeably used. Similarly WSH_DEREGISTER_NAME and WSH_REMOVE_NAME.

```
int PASCAL FAR setsockopt (  
    SOCKET s,  
    int level,  
    int optname,  
    char FAR * optval,  
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_DEREGISTER_NAME
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_DEREGISTER_NAME.
<i>optlen</i>	The size of the <i>optval</i> buffer (sizeof (WSH_DEREGISTER_NAME)).
Remarks	This will deregister the name specified in the structure on the bound socket specified. If the name is not currently registered, this call will return a success code.

Network Information

There is no network activity.

Error Codes

No special error codes are defined.

Sample Code

```
WSH_DEREGISTER_NAME  regName;  
  
// SETSOCKOPT (register name)  
regName.ObjectNameLen = strlen("Windows Socket Server");  
regName.TypeNameLen = strlen("Windows Socket Type");  
regName.ZoneNameLen = strlen("BLDG1 Zone");  
strcpy(regName.ObjectName, "Windows Socket Server");  
strcpy(regName.TypeName, "Windows Socket Type");  
strcpy(regName.ZoneName, "BLDG1 Zone");  
  
bytesWritten = sizeof(WSH_DEREGISTER_NAME);  
  
err = setsockopt( listenHandle, SOL_APPLETALK,  
                 SO_DEREGISTER_NAME, (char *)&regName, bytesWritten );  
  
if ( err == SOCKET_ERROR )  
{  
    printf( "failed: %ld\n", GetLastError( ));  
}
```

2.5.3 SO_LOOKUP_NAME

```
int PASCAL FAR getsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR * optval,
    int optlen );
```

s A descriptor identifying a socket on which the name is to be deregistered.

level The level at which the option is defined; must be SOL_APPLETALK.

optname Must be SO_LOOKUP_NAME

optval A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_LOOKUP_NAME.

optlen The size of the *optval* buffer (sizeof (WSH_LOOKUP_NAME) + sizeof(number of tuples)). There must be space for atleast one tuple following the WSH_LOOKUP_NAME structure.

Remarks Looks up a specified NBP name (may contain wildcards) and returns the matching tuples of names and nbp information including addresses in the buffer specified. The WSH_LOOKUP_NAME structure is defined as:

```
typedef struct _WSH_LOOKUP_NAME
{
    WSH_NBP_TUPLE      LookupTuple;
    ULONG              NoTuples;

    //
    // Array of NoTuples WSH_NBP_TUPLE structures
    //
} WSH_LOOKUP_NAME, *PWSH_LOOKUP_NAME;
```

Network Information

On a routed network upto ten NBP broadcast request packets are sent to a-router spaced one second apart till a NbpResponse packet is received. On a nonrouted network lookup requests are sent instead of broadcast requests.

Error Codes

WSAENOBUFFS The buffer was too small to hold all tuples received.

Sample Code

```
UCHAR          lookupBuffer[1024];
PWSH_LOOKUP_NAME lookupName;
PWSH_NBP_TUPLE tuple;

lookupName = (PWSH_LOOKUP_NAME)lookupBuffer;
bytesWritten = sizeof(lookupBuffer);
```

```

lookupName->LookupTuple.NbpName.ObjectNameLen =
    strlen(DGRAM_SERVER_NAME);
lookupName->LookupTuple.NbpName.TypeNameLen =
    strlen(CLIENT_LOOKUPTYPE);
lookupName->LookupTuple.NbpName.ZoneNameLen =
    strlen(CLIENT_LOOKUPZONE);

strcpy(lookupName->LookupTuple.NbpName.ObjectName, DGRAM_SERVER_NAME);
strcpy(lookupName->LookupTuple.NbpName.TypeName, CLIENT_LOOKUPTYPE);
strcpy(lookupName->LookupTuple.NbpName.ZoneName, CLIENT_LOOKUPZONE);

err = getsockopt( handle, SOL_APPLETALK, SO_LOOKUP_NAME,
    (char *)lookupBuffer, &bytesWritten );

// Number of tuples, we print out only the first tuple.
i = ((PWSH_LOOKUP_NAME)lookupBuffer)->NoTuples;
printf("\nNumTuples %ld\n", i);

tuple = (PWSH_NBP_TUPLE)(lookupBuffer+sizeof(WSH_LOOKUP_NAME));
if (i > 0)
{
    object = tuple->NbpName.ObjectName;
    for (j=0; j < tuple->NbpName.ObjectNameLen; j++)
        printf("%c", object[j]);

    printf(":" );
    type = tuple->NbpName.TypeName;
    for (j=0; j < tuple->NbpName.TypeNameLen; j++)
        printf("%c", type[j]);

    printf("@");
    zone = tuple->NbpName.ZoneName;
    for (j=0; j < tuple->NbpName.ZoneNameLen; j++)
        printf("%c", zone[j]);

    printf(" - ");

    printf("%lx.%lx.%lx\n",
        tuple->Address.Network,
        tuple->Address.Node,
        tuple->Address.Socket);
}

```

2.5.4 SO_CONFIRM_NAME

```
int PASCAL FAR getsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR * optval,
    int optlen );
```

s A descriptor identifying a socket on which the name is to be deregistered.

level The level at which the option is defined; must be SOL_APPLETALK.

optname Must be SO_CONFIRM_NAME

optval A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_NAME_TUPLE.

optlen The size of the *optval* buffer (sizeof (WSH_NBP_TUPLE)).

Remarks This will confirm that the name specified is bound to the same address specified.

Network Information

A NBP Lookup Request is sent to the address to be confirmed.

Error Codes	WSAEADDRNOTAVAIL	The address is
	confirmed with a new socket.	

Sample Code

```
WSH_NBP_TUPLE            tuple;
WSH_ATALK_ADDRESS        confirmAddr;

confirmAddr = (Address from a previous lookup of the following name)

bytesWritten = sizeof(WSH_NBP_TUPLE);
tuple.NbpName.ObjectNameLen =
    strlen(DGRAM_SERVER_NAME);
tuple.NbpName.TypeNameLen =
    strlen(CLIENT_LOOKUPTYPE);
tuple.NbpName.ZoneNameLen =
    strlen(CLIENT_LOOKUPZONE);

strcpy(tuple.NbpName.ObjectName, DGRAM_SERVER_NAME);
strcpy(tuple.NbpName.TypeName, CLIENT_LOOKUPTYPE);
strcpy(tuple.NbpName.ZoneName, CLIENT_LOOKUPZONE);

tuple.Address = confirmAddress;
err = getsockopt( handle, SOL_APPLETALK,
    SO_CONFIRM_NAME, (char *)&tuple, &bytesWritten );

printf("%lx.%lx.%lx\n", tuple->Address.Network, tuple->Address.Node,
    tuple->Address.Socket);
```

2.5.5 SO_LOOKUP_MYZONE, SO_GETMYZONE

SO_LOOKUP_MYZONE and SO_GETMYZONE can be inter-changeably used.

```
int PASCAL FAR getsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR * optval,
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_LOOKUP_MYZONE
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied.
<i>optlen</i>	The size of the <i>optval</i> buffer (must be atleast MAX_ENTITY_LEN+1 (= 33)).

Remarks This will return the default zone on the net.

Network Information

A ZIP GetMyZone request is sent.

Error Codes

No special error codes are defined.

Sample Code

```
char  buffer[MAX_ENTITY_LEN+1];
bytesWritten = sizeof(buffer);

err = getsockopt( handle, SOL_APPLETALK, SO_LOOKUP_MYZONE, (char *)buffer
                  &bytesWritten );

printf("%s", buffer);

if ( err == SOCKET_ERROR ) {
    printf( "failed: %ld (%lx)\n", GetLastError( ), GetLastError( ) );
    return;
}
```

2.5.6 SO_LOOKUP_ZONES, SO_GETZONELIST

SO_LOOKUP_ZONES and SO_GETZONELIST can be interchangeably used.

```
int PASCAL FAR getsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR * optval,
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_LOOKUP_ZONES
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_LOOKUP_ZONES.
<i>optlen</i>	The size of the <i>optval</i> buffer (sizeof (WSH_LOOKUP_ZONES)).
Remarks	Upon return from the call, the space after the WSH_LOOKUP_ZONES is filled in with NULL terminated zone names (ANSI). This call returns the internet zone list. The number of zones found and present in the buffer is indicated by NoZones. This call will fail with an error if there is not atleast one byte following the NoZones field for the implied Zones buffer.

Network Information

A ZIP GetZoneList request is sent.

Error Codes

No special error codes are used.

Sample Code

```
char buffer[1024];

lookupZones = (PWSH_LOOKUP_ZONES)buffer;
bytesWritten = sizeof(buffer);

err = getsockopt( handle, SOL_APPLETALK,
                  SO_LOOKUP_ZONES, (char *)buffer, &bytesWritten );

// Number of zones
i = ((PWSH_LOOKUP_ZONES)buffer)->NoZones;
printf("\nNumZones %ld\n", i);

zone = buffer+sizeof(WSH_LOOKUP_ZONES);
while (i-- > 0)
{
    printf("%s ", zone);
    zone = zone + strlen(zone) + 1;
}
```

2.5.7 SO_LOOKUP_ZONES_ON_ADAPTER, SO_GETLOCALZONES

SO_LOOKUP_ZONES_ON_ADAPTER and SO_GET_LOCAL_ZONES can be inter-changeably used.

```
int PASCAL FAR getsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR * optval,
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_LOOKUP_ZONES.
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_LOOKUP_ZONES. Following the structure should be the WCHAR string with the name on the adapter on which the local zones are to be obtained.
<i>optlen</i>	The size of the <i>optval</i> buffer (sizeof (WSH_LOOKUP_ZONES)).

Remarks Upon return from the call, the space after the WSH_LOOKUP_ZONES is filled in with NULL terminated zone names (ANSI). The number of zones found and present in the buffer is indicated by NoZones. This call will fail with an error if there is not atleast one byte following the NoZones field for the implied Zones buffer. The adapter name is passed in after the structure and is overwritten by the zone names upon return.

Network Information

A ZIP GetLocalZones packet is sent out on the net of the adapter indicated.

Error Codes

No special error codes are used.

Sample Code

```
#define ADAPTER_NAME L"\\Device\\Elnkii01"

lookupZones = (PWSH_LOOKUP_ZONES)buffer;
bytesWritten = sizeof(buffer);

// Put in the adapter name.
wcscpy((lookupZones+1), ADAPTER_NAME);
err = getsockopt( handle, SOL_APPLETALK, SO_LOOKUP_ZONES_ON_ADAPTER,
    (char *)buffer, &bytesWritten );

// Number of zones
i = ((PWSH_LOOKUP_ZONES)buffer)->NoZones;
printf("NumZones %ld\n", i);
zone = buffer+sizeof(WSH_LOOKUP_ZONES);
while (i-- > 0)
{
    printf("%s ", zone);
    zone = zone + strlen(zone) + 1;
}
```


2.5.8 SO_LOOKUP_NETDEF_ON_ADAPTER, SO_GETNETINFO

SO_LOOKUP_NETDEF_ON_ADAPTER and SO_GETNETINFO can be inter-changeably used.

```
int PASCAL FAR setsockopt (  
    SOCKET s,  
    int level,  
    int optname,  
    char FAR * optval,  
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_LOOKUP_NETDEF_ON_ADAPTER.
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_LOOKUP_NETDEF_ON_ADAPTER.
<i>optlen</i>	The size of the <i>optval</i> buffer (sizeof (WSH_LOOKUP_NETDEF_ON_ADAPTER)).
Remarks	This will return the seeded values for the network numbers and an ansi string for the default zone for the net on the indicated adapter. The adapter is passed in as a WCHAR string following the structure and is overwritten by the default zone returned. If the network is not seeded, then the network range of 1-0xFFFE is returned and the zone list has a single zone “*”.

Network Information

A ZIP GetNetInfo packet is sent out on the adapter indicated.

Error Codes

No special error codes are used.

Sample Code

```
#define      ADAPTER_NAME      L"\\Device\\Elnkii01"  
  
lookupInfo = (PWSH_LOOKUP_NETDEF_ON_ADAPTER)buffer;  
bytesWritten = sizeof(buffer);  
  
// Put in the adapter name.  
wcscpy((lookupInfo+1), ADAPTER_NAME);  
err = getsockopt(handle, SOL_APPLETALK,  
    SO_LOOKUP_NETDEF_ON_ADAPTER,  
    (char *)buffer, &bytesWritten );  
  
// Net range  
printf("Netnum low/high %ld.%ld\n",  
    lookupInfo->NetworkRangeLowerEnd, lookupInfo->NetworkRangeUpperEnd);  
  
zone = buffer+sizeof(WSH_LOOKUP_NETDEF_ON_ADAPTER);  
printf("default zone: %s ", zone);
```

2.5.9 SO_PAP_SET_SERVER_STATUS

```
int PASCAL FAR setsockopt (
    SOCKET s,
    int level,
    int optname,
    char FAR * optval,
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_PAP_SET_SERVER_STATUS.
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. This will just be a buffer of atmost 255 bytes containing the status to be associated with the socket.
<i>optlen</i>	The size of the <i>optval</i> buffer (<= 255 bytes).
Remarks	This will set the status as the status to be sent out if a remote client sends a PAP get status packet. If this call passes in a NULL optval, then the previously set status is erased.

Network Information

There is no network activity.

Error Codes

No special error codes are used.

Sample Code

```
char        status[255];

err = setsockopt( handle, SOL_APPLETALK,
                  SO_PAP_SET_SERVER_STATUS,
                  status,
                  sizeof(status));

if ( err == SOCKET_ERROR ) {
    printf( "failed: %ld (%lx)\n", GetLastError( ), GetLastError( ) );
}
```

2.5.10 SO_PAP_GET_SERVER_STATUS

```
int PASCAL FAR getsockopt (  
    SOCKET s,  
    int level,  
    int optname,  
    char FAR * optval,  
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_PAP_GET_SERVER_STATUS.
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. Must be of type WSH_PAP_GET_SERVER_STATUS.
<i>optlen</i>	The size of the <i>optval</i> buffer (sizeof (WSH_PAP_GET_SERVER_STATUS)).

Remarks This will get the PAP status registered on the address specified in ServerAddr (which generally would have been determined via NbpLookup). The four reserved bytes will correspond to the four reserved bytes in the PAP status packet. These will be in the network format.

Network Information

A PAP GetStatus packet is sent to the indicated address.

Error Codes No special error codes are used.

Sample Code

```
WSH_PAP_GET_SERVER_STATUS      getStatus;  
  
getStatus.ServerAddr.sat_family = ... // Set the address of the PAP server  
  
err = getsockopt( handle, SOL_APPLETALK,  
                  SO_PAP_GET_SERVER_STATUS, (char *)&getStatus,  
                  sizeof(getStatus));  
if (err > 0)  
{  
    // status returned in getStatus.ServerStatus with the leading byte indicating  
    the length of the          // status string.  
}
```

2.5.11 SO_PAP_PRIME_READ

```
int PASCAL FAR setsockopt (  
    SOCKET s,  
    int level,  
    int optname,  
    char FAR * optval,  
    int optlen );
```

<i>s</i>	A descriptor identifying a socket on which the name is to be deregistered.
<i>level</i>	The level at which the option is defined; must be SOL_APPLETALK.
<i>optname</i>	Must be SO_PRIMEREAD
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied. Must be a buffer which will be used for a subsequent recv().
<i>optlen</i>	The size of the <i>optval</i> buffer (MUST be MIN_PAP_READ_BUF_SIZE (4Kbytes)).
Remarks	This will prime a read on the PAP connection. This enables the remote client to send the data, without the local client having done a recv(). Then when the data comes in, the select() call is triggered for read, and the client can do a recv() with the same buffer that was passed into the PrimeRead call. Upon return from the recv() the received data will be in the buffer. This allows support for non-blocking sockets on the read-driven PAP protocol. Note that receive should be followed by another setsockopt() for SO_PAP_PRIME_READ. This is not implicit.

Network Information

A PAP SendData is sent to the remote end. The remote may then send PAP data packets.

Error Codes

No special error codes are used.

Sample Code

```
unsigned char *pBuffer;  
  
if ((pBuffer = RtlAllocateHeap(RtlProcessHeap(), 0, MIN_PAP_READ_BUF_SIZE)) == NULL)  
    return;  
  
err = setsockopt(handle, SOL_APPLETALK, SO_PAP_PRIME_READ, pBuffer,  
                MIN_PAP_READ_BUF_SIZE);  
  
// If no error, go ahead and do a select() on readfds. When it is triggered, do a recv()  
// passing in the same  
// buffer passed into PrimeRead.  
err = WsaRecvEx(handle, pBuffer, MIN_PAP_READ_BUF_SIZE, &flags);
```

2.6 Database Files

These files are not used in Appletalk. Instead, the functionality provided by the Name Binding Protocol primitives (provided as socket options) should be used for name binding and resolution.

2.7 Maximum number of sockets supported

There are a certain maximum number of Appletalk sockets available for use. Even if the limit is not hit for the Windows Sockets, it may hit the underlying protocol limit. The Windows NT implementation of the Appletalk protocol creates two Appletalk nodes on an extended port, thus doubling the number of dynamic sockets available for use. The exact number of available sockets will depend on other applications running on the system (File Server, Print Server, Appletalk Monitor ports etc.).

3. SOCKET LIBRARY OVERVIEW

3.1 Socket Functions

The Windows Sockets specification includes the following Berkeley-style socket routines:

accept()

An incoming connection is acknowledged and associated with an immediately created socket. The original socket is returned to the listening state. Note semantics for PAP and ADSP.

bind()

Assign a local name to an unnamed socket. Creates an Appletalk endpoint and associates it with the Winsock socket.

closesocket()

Remove a socket descriptor from the per-process object reference table. It should be noted that this will perform a disconnect (abortive) of an existing connection for an ADSP/PAP socket.

connect()

Initiate a connection on the specified socket.

getpeername()

Retrieve the name of the peer connected to the specified socket descriptor.

getsockname()

Retrieve the current name for the specified socket

getsockopt()

Retrieve options associated with the specified socket descriptor. Also used for the special Appletalk options.

htonl()

Convert a 32-bit quantity from host byte order to network byte order.

htons()

Convert a 16-bit quantity from host byte order to network byte order.

inet_addr()

Not used in Appletalk.

inet_ntoa()

Not used in Appletalk.

ioctlsocket()

Provide control for descriptors.

listen()

Listen for incoming connections on a specified socket.

ntohl()

Convert a 32-bit quantity from network byte order to host byte order.

ntohs()

Convert a 16-bit quantity from network byte order to host byte order.

recv()

Receive data from a connected socket. Check notes for PAP.

recvfrom()

Receive data from either a connected or unconnected socket.

select()

Perform synchronous I/O multiplexing.

send()

Send data to a connected socket.

sendto()

Send data to either a connected or unconnected socket.

setsockopt()

Store options associated with the specified socket descriptor. Also used for the special Appletalk options.

shutdown()

For Appletalk, this call will shutdown the connection in both directions.

socket()

Create an endpoint for communication and return a socket descriptor.

4. SOCKET LIBRARY REFERENCE

4.1 Socket Routines

This chapter presents the socket library routines in alphabetical order, and describes any differences from the Windows Sockets 1.1 specification for Appletalk.

4.1.1 accept()

Description Accept a connection on a socket.

#include <winsock.h>

SOCKET PASCAL FAR accept (**SOCKET** *s*, **struct sockaddr FAR *** *addr*,
int FAR * *addrlen*);

s A descriptor identifying a socket which is listening for connections after a **listen()**.

addr The address of the connecting entity, as known to the communications layer. This will return the net/node/socket of the remote address.

addrlen A pointer to an integer which contains the length of the address *addr*.

Remarks This call needs to be made to get a socket descriptor to perform activity on the established connection. Note that the Appletalk connection has already been made at this point.

Network Information

There is no network activity.

Return Value As in Windows Sockets 1.1 specification.

As in Windows Sockets 1.1 specification.

4.1.2 bind()

Description Associate a local address with a socket.

#include <winsock.h>

int PASCAL FAR bind (SOCKET *s*, struct sockaddr FAR * *name*, int *namelen*);

s A descriptor identifying an unbound socket.

name The address to assign to the socket. The sockaddr structure is defined as follows:

```
struct sockaddr_at {  
    u_short sat_family;  
    u_short      sat_net;  
    u_char      sat_node;  
    u_char      sat_socket;  
} SOCKADDR_AT, *PSOCKADDR_AT;
```

namelen The length of the *name*.

Remarks This routine will create an Appletalk socket endpoint in the transport. The sat_net/sat_node values are ignored. If sat_socket has a value of 0, then it is assumed the application wants to open a dynamic socket and a socket in the range 0x80 - 0xFE is opened. Values from 0x01 - 0x7F will open a static socket (i.e. socket number will be that which is specified in the call). Note that among these, some socket numbers are reserved, and these could fail with an error.

Network Information

There is no network activity.

Return Value If no error occurs, **bind()** returns 0. Otherwise, it returns SOCKET_ERROR, and a specific error code may be retrieved by calling **WSAGetLastError()**.

As in Windows Sockets 1.1 specification.

4.1.3 closesocket()

Description Close a socket.

```
#include <winsock.h>
```

```
int PASCAL FAR closesocket ( SOCKET s );
```

s A descriptor identifying a socket.

Remarks This will close any associated connections abortively. The Appletalk socket is closed.

Return Value If no error occurs, **closesocket()** returns 0. Otherwise, a value of **SOCKET_ERROR** is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

As in Windows Sockets 1.1 specification.

4.1.4 connect()

Description Establish a connection to a peer.

```
#include <winsock.h>
```

```
int PASCAL FAR connect ( SOCKET s, struct sockaddr FAR * name,  
int namelen );
```

s A descriptor identifying an unconnected socket.

name The name of the peer to which the socket is to be connected.

namelen The length of the *name*.

Remarks If the `sa_socket` field is 0, or if the socket is not bound, **connect()** will return the error `WSAEINVAL`. There is no automatic binding capability in Appletalk, the application must explicitly do so before using the socket in a **connect()** call.

Network Information

For ADSP, an `OpenConnectionRequest` is sent on the net. For PAP, an `OpenConn` request is sent out on the net. There is no network activity for DDP.

Return Value If no error occurs, **connect()** returns 0. Otherwise, it returns `SOCKET_ERROR`, and a specific error code may be retrieved by calling **WSAGetLastError()**.

4.1.5 getpeername()

Description Get the address of the peer to which a socket is connected.

```
#include <winsock.h>
```

```
int PASCAL FAR getpeername ( SOCKET s, struct sockaddr FAR * name, int FAR  
* namelen );
```

s A descriptor identifying a connected socket.

name The structure which is to receive the name of the peer.

namelen A pointer to the size of the *name* structure.

Remarks As in Windows Sockets 1.1. specification.

Return Value If no error occurs, **getpeername()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

4.1.6 getsockname()

Description Get the local name for a socket.

```
#include <winsock.h>
```

```
int PASCAL FAR getsockname ( SOCKET s, struct sockaddr FAR * name,  
int FAR * namelen );
```

s A descriptor identifying a bound socket.

name The name of the socket.

namelen The size of the *name* array.

Remarks As in Windows Sockets 1.1 specification.

Return Value If no error occurs, **getsockname()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

4.1.7 getsockopt()

Description Retrieve a socket option.

#include <winsock.h>

**int PASCAL FAR getsockopt (SOCKET *s*, int *level*, int *optname*,
char FAR * *optval*, int FAR * *optlen*);**

<i>s</i>	A descriptor identifying a socket.
<i>level</i>	The level at which the option is defined; the only supported <i>level</i> is SOL_SOCKET.
<i>optname</i>	The socket option for which the value is to be retrieved.
<i>optval</i>	A pointer to the buffer in which the value for the requested option is to be returned.
<i>optlen</i>	A pointer to the size of the <i>optval</i> buffer.

Remarks In addition to the options specified in Windows Sockets 1.1, some special options are available for sockets using the Appletalk protocols. See section describing the socket options for usage.

Calling **getsockopt()** with an unsupported option will result in an error code of WSAENOPROTOOPT being returned from **WSAGetLastError()**.

Return Value If no error occurs, **getsockopt()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

Error Codes See section on socket options for any specific error codes.

4.1.8 htonl()

Description Convert a **u_long** from host to network byte order.

#include <winsock.h>

u_long PASCAL FAR htonl (u_long *hostlong*);

hostlong A 32-bit number in host byte order.

Remarks As in Windows Sockets 1.1 specification.

4.1.9 htons()

Description Convert a **u_short** from host to network byte order.

```
#include <winsock.h>
```

```
u_short PASCAL FAR htons ( u_short hostshort );
```

hostshort A 16-bit number in host byte order.

Remarks As in Windows Sockets 1.1 specification.

4.1.10 inet_addr()

Description Convert a string containing a dotted address into an **in_addr**.

```
#include <winsock.h>
```

```
unsigned long PASCAL FAR inet_addr ( char FAR * cp );
```

cp A character string representing a number expressed in the Internet standard "." notation.

Remarks Not used for Appletalk.

4.1.11 inet_ntoa()

Description Convert a network address into a string in dotted format.

```
#include <winsock.h>
```

```
char FAR * PASCAL FAR inet_ntoa ( struct in_addr in );
```

in A structure which represents an Internet host address.

Remarks Not used for Appletalk.

4.1.12 ioctlsocket()

Description Control the mode of a socket.

```
#include <winsock.h>
```

```
int PASCAL FAR ioctlsocket ( SOCKET s, long cmd, u_long FAR * argp );
```

s A descriptor identifying a socket.

cmd The command to perform on the socket *s*.

argp A pointer to a parameter for *cmd*.

Remarks As in Windows Sockets 1.1 specification.

4.1.13 listen()

Description Establish a socket to listen for incoming connection.

```
#include <winsock.h>
```

```
int PASCAL FAR listen ( SOCKET s, int backlog );
```

s A descriptor identifying a bound, unconnected socket.

backlog The maximum length to which the queue of pending connections may grow.

Remarks As in Windows Sockets 1.1 specification.

4.1.14 ntohs()

Description Convert a **u_long** from network to host byte order.

```
#include <winsock.h>
```

```
u_long PASCAL FAR ntohs ( u_long netlong );
```

netlong A 32-bit number in network byte order.

Remarks As in Windows Sockets 1.1 specification.

4.1.15 ntohs()

Description Convert a **u_short** from network to host byte order.

```
#include <winsock.h>
```

```
u_short PASCAL FAR ntohs ( u_short netshort );
```

netshort A 16-bit number in network byte order.

Remarks As in Windows Sockets 1.1. specification.

4.1.16 recv()/WsaRecvEx()

Description Receive data from a socket.

#include <winsock.h>

int PASCAL FAR recv (**SOCKET** *s*, **char FAR *** *buf*, **int** *len*, **int** *flags*);

s A descriptor identifying a connected socket.

buf A buffer for the incoming data.

len The length of *buf*.

flags Specifies the way in which the call is made.

Remarks

This function takes on different semantics only for non-blocking sockets on PAP. For a blocking socket, a posted recv() will complete with an error code (WSAEPENDING) if the receive is posted successfully with the Appletalk transport. For non-blocking sockets the socket option SO_PAP_PRIME_READ needs to happen to enable the remote end to send data. Then a recv() is done with the same buffer used in prime read when select() indicates data is to be read. After a recv() consumes the data another prime read should be posted.

PAP will not support the MSG_PEEK or the MSG_OOB flags. Also, the receive buffer posted for PAP must be the maximum expected size of the data (value is 4Kbytes).

WsaRecvEx() takes a int * for the flags and returns the flags indicating the type of data. For recv() if a partial message is received, then a negative value is returned as the result with the last error being set to WSAEMSGPARTIAL. Flipping the sign of the result then gives the number of bytes actually received. It is recommended that WsaRecvEx() be used when EOM indications are to be used.

Error Codes

As in Windows Sockets 1.1 specification.

4.1.17 recvfrom()

Description Receive a datagram and store the source address.

#include <winsock.h>

int PASCAL FAR recvfrom (**SOCKET** *s*, **char FAR *** *buf*, **int** *len*, **int** *flags*,
struct sockaddr FAR * *from*, **int FAR *** *fromlen*);

s A descriptor identifying a bound socket.

buf A buffer for the incoming data.

len The length of *buf*.

flags Specifies the way in which the call is made.

from Points to a buffer which will hold the source address upon return.

fromlen A pointer to the size of the *from* buffer.

Remarks For DDP, only those datagrams whose protocol type matches the protocol type set on the socket during the socket call can be received.

Error Codes As in Windows Sockets 1.1 specification.

4.1.18 select()

Description Determine the status of one or more sockets, waiting if necessary.

#include <winsock.h>

int PASCAL FAR select (int *nfds*, fd_set FAR * *readfds*, fd_set FAR * *writefds*, fd_set FAR * *exceptfds*, struct timeval FAR * *timeout*);

<i>nfds</i>	This argument is ignored and included only for the sake of compatibility.
<i>readfds</i>	A set of sockets to be checked for readability.
<i>writefds</i>	A set of sockets to be checked for writeability
<i>exceptfds</i>	A set of sockets to be checked for errors.
<i>timeout</i>	The maximum time for select() to wait, or NULL for blocking operation.

Remarks As in Windows Sockets 1.1 specification.

Error Codes As in Windows Sockets 1.1 specification.

4.1.19 send()

Description Send data on a connected socket.

#include <winsock.h>

int PASCAL FAR send (SOCKET *s*, char FAR * *buf*, int *len*, int *flags*);

s A descriptor identifying a connected socket.

buf A buffer containing the data to be transmitted.

len The length of the data in *buf*.

flags Specifies the way in which the call is made.

Remarks For datagram sockets, the protocol type sent with the datagram is that set on the socket initially during creation.

Error Codes As in Windows Sockets 1.1. specification.

4.1.20 sendto()

Description Send data to a specific destination.

#include <winsock.h>

**int PASCAL FAR sendto (SOCKET *s*, char FAR * *buf*, int *len*, int *flags*,
struct sockaddr FAR * *to*, int *tolen*);**

<i>s</i>	A descriptor identifying a socket.
<i>buf</i>	A buffer containing the data to be transmitted.
<i>len</i>	The length of the data in <i>buf</i> .
<i>flags</i>	Specifies the way in which the call is made.
<i>to</i>	A pointer to the address of the target socket.
<i>tolen</i>	The size of the address in <i>to</i> .

Remarks For DDP, the protocol type sent with the DDP datagram is that initially set on the socket during creation.

Error Codes As in Windows Sockets 1.1 specification.

4.1.21 setsockopt()

Description Set a socket option.

```
#include <winsock.h>
```

```
int PASCAL FAR setsockopt ( SOCKET s, int level, int optname,  
char FAR * optval, int optlen );
```

<i>s</i>	A descriptor identifying a socket.
<i>level</i>	The level at which the option is defined; the only supported <i>level</i> is SOL_SOCKET.
<i>optname</i>	The socket option for which the value is to be set.
<i>optval</i>	A pointer to the buffer in which the value for the requested option is supplied.
<i>optlen</i>	The size of the <i>optval</i> buffer.

Remarks See the socket options section for all extra options for Appletalk.

Return Value If no error occurs, **setsockopt()** returns 0. Otherwise, a value of SOCKET_ERROR is returned, and a specific error code may be retrieved by calling **WSAGetLastError()**.

Error Codes See section on socket options for specific error codes.

4.1.22 shutdown()

Description Disable sends and/or receives on a socket.

```
#include <winsock.h>
```

```
int PASCAL FAR shutdown ( SOCKET s, int how );
```

s A descriptor identifying a socket.

how A flag that describes what types of operation will no longer be allowed.

Remarks For the Appletalk protocols, **shutdown()** will abortively disconnect active connections. There is no facility for a graceful shutdown.

Error Codes As in Windows Sockets 1.1 specification.

4.1.23 socket()

Description Create a socket.

```
#include <winsock.h>
```

```
SOCKET PASCAL FAR socket ( int af, int type, int protocol );
```

af An address format specification. For Appletalk, the PF_APPLETALK address format is used.

type A type specification for the new socket.

protocol A particular protocol to be used with the socket.

Remarks For valid protocol types etc., please see the beginning of this document.

Error Codes As in Windows Sockets 1.1 specification.

Appendix A. Error Codes and Header Files

A.1 Error Codes

The following is a list of possible error codes returned by the **WSAGetLastError()** call, along with their explanations, that are specific to Appletalk.

<u>Windows Sockets Code</u>	<u>Berkeley Equivalent</u>	<u>Error</u>	<u>Interpretation</u>
WSAEMSGPARTIAL	-	10100	A partial message was received.

A.2 Header Files

A.2.1 Appletalk Header File

The following file will need to be included by an application developer writing for Appletalk over Windows Sockets.

ATALKWSH.H

Appendix B. Notes for Windows Sockets For Appletalk Users

B.1 Introduction

In addition to all the Windows Sockets components, a user will also need a helper DLL which implements all the Appletalk specific options.

B.2 Windows Sockets For Appletalk Components

<u>Component</u>	<u>Description</u>
SFMWSHAT.DLL	The Windows Sockets For Appletalk Helper DLL