Proposed Sample API for the 1394PWG Transport Protocol

Brian Batchelder

Revision 0.2

April 13, 1999

1 Introduction

The 1394PWG Transport Protocol is a general-purpose transport protocol for the IEEE 1394 link. The transport protocol:

- supports multiple, concurrent, independent, symmetrical connections;
- provides in-order, byte-stream and in-order, datagram services;
- provides a directory service;
- transparently handles transient link interruptions;
- provides out-of-band (*are we absolutely sure we need this???*) and end-of-message indications.

In addition, the transport protocol:

- is data, application and O/S independent;
- does not preclude concurrent operation of other protocol stacks.

The details of these capabilities are covered in other documents.

This document presents a proposal for a sample API for clients of the 1394PWG transport protocol. Work on this proposal will produce an *informative* annex for the 1394PWG transport protocol specification. The API will not be mandated. 1394PWG transport protocol implementations are free to create other APIs.

The 1394PWG Sample API is a socket API, in which the client and transport use sockets as a handle for a connection. For this document, the overall API has been grouped into three sub-APIs: the Connect API, the Data Transfer API and the Disconnect API. Clients create sockets and make connections through them to remote services using entry points in the Connect API. Clients transfer data using entry points in the Data Transfer API. Clients close connections and release sockets using entry points in the Disconnect API.

This sample API is modeled after the Winsock-2 API, available at http://www.sockets.com. The critical entry points are summarized in this document. More detail on each of these entry points and many other interesting entry points are available in the Winsock-2 API specification.

Add non-blocking sockets.

1.1 Special Data

1.1.1 Out of band data

Data can be marked "out-of-band." The 1394PWG transport shall process out-of-band data in the same manner as in-band data. In-band data shall not be combined with out-of-band data. The receiver of out-of-band data shall be notified that the data is out-of-band.

1.1.2 Messages

Data in the data stream can be grouped into messages. The last buffer of data in the message is marked "end of message." The 1394PWG transport shall process end-of-message data in the same manner as non end-of-message data. End-of-message data shall not be combined with non end-of-message data. The receiver of end-of-message data shall be notified that the data is the end of a message.

1.2 Definitions

Graceful Disconnect: In a graceful disconnect sequence, any data that has been queued but not yet transmitted shall be sent prior to the connection being closed. To execute a graceful disconnect, the client issues shutdown(), followed by closesocket() after all data has been transferred.

Abortive Disconnect: In an abortive shutdown, any unsent data is lost. To execute an abortive disconnect, the client issues closesocket() either:

a) without issuing shutdown() or
b) after issuing shutdown() but before remote client has issued shutdown().

2 Connect API

Transport clients use the Connect API to create connections to other transport clients.

It provides entries to:

- create sockets;
- bind service names to sockets;
- listen for and accept connections on sockets;
- connect to other clients;
- modify the operation of the connections;
- request asynchronous notifications of activity on the connection.

2.1 socket()

Socket establishes a new endpoint. The 1394PWG transport layer manages the allocation of sockets. Neither clients nor servers need be aware of the management of sockets.

***socket* = socket (**
IN ***address_family***
IN ***socket_type,***
IN ***protocol*)**

socket

A new socket allocated by the 1394PWG transport layer. If there is an error, INVALID_SOCKET is returned.

address_family

An address family specification. This is used to differentiate between the 1394PWG transport and other transport protocols (e.g., TCP/IP). There is currently no address family specification for the 1394PWG transport. One can be requested from the Winsock-2 group.

socket_type

A type specification for the new socket.

The following is the only *socket_type* supported for the 1394PWG transport:

- SOCK_STREAM: Provides connection-based byte streams with out-of-band data transmission and message-indication mechanisms. The transport is free to divide or aggregate client buffers. In-band data shall not be combined with out-of-band data. End-of-message data shall not be combined with non end-of-message data.

protocol

Protocol to be used. Must be support on *address_family*.

2.2 bind()

Bind assigns a name to a socket. A server uses bind to register its name so that it can be found through service discovery. Bind will fail if *service_name* is already bound to another socket.

bind (
IN ***socket,***
IN ***name,***
IN ***namelength*)**

socket
A socket previously allocated by a call to socket().

name
Address family-dependent address and service name of the device and service to which the connection is to be opened. For the 1394PWG Transport Protocol, *name* contains the local memory address of the local 1394PWG SBP-2 unit and the service name. To support hosts without unit directories, if the memory address is NULL, connections will be accepted across any 1394PWG SBP-2 login from the host. Service names may be up to 40 characters taken from the set of uppercase letters, digits, and the punctuation character hyphen. They must start with a letter, and end with a letter or digit. Service names may be registered in the IANA Protocol and Service Names list, available at http://www.iana.org.

namelength
Length of *name.*

2.3 listen()

Listen allocates space for queuing incoming connection requests.

listen (
IN ***socket***
IN ***backlog*)**

socket
A socket previously allocated by a call to socket() and assigned a name or explicit socket number by a call to bind().

backlog
The maximum length to which the queue of pending connections may grow. If this value is SOMAXCONN, then the underlying service provider responsible for *socket* will set the backlog to a maximum "reasonable" value. Listen() may be again to change *backlog.*

2.4 accept()

Accept() accepts new connections on a socket. When a connect request is received for that socket from the remote transport, the 1394PWG transport layer allocates a new socket for that connection and the call to accept() completes.

***new_socket* = accept (**
IN ***socket,***
OUT ***address,***
IN OUT ***addresslength*)**

new_socket
A new socket allocated for this connection by the 1394PWG transport layer. Communication on this connection references *new_socket.*

socket
A socket previously allocated by a call to socket(), assigned a name by a call to bind(), and placed in a listen state by a call to listen(). After accept() returns, *socket* may be used in further accept() calls to allow other connections to this service.

address
An optional pointer to a buffer which receives the address of the connecting entity, as known to the communications layer. The exact format of the address is determined by the address family established when *socket* was created.

addresslength
An optional pointer to an integer which contains the length of *address*.

2.5 connect()

Connect() opens a connection to a server.

connect (
IN ***socket,***
IN ***name,***
IN ***namelength*)**

socket
A socket previously allocated by a call to socket().

name
Address family-dependent address and service name of the device and service to which the connection is to be opened. For the 1394PWG Transport Protocol, *name* contains the memory address of the 1394PWG SBP-2 unit (or beginning of the device's config ROM, if there is no 1394PWG SBP-2 unit) and the service name.

namelength
Length of *name*.

2.6 setsockopt()

Setsockopt() sets socket options. Since the client is not required to call setsockopt(), the implementation shall establish reasonable default values for each of the socket options.

setsockopt (

IN	***socket,***
IN	***level,***
IN	***optionname,***
IN	***optionvalue,***
IN	***optionlength*****)**

socket
A socket previously allocated by a call to socket() or accept().

level
Which level of the stack defined the option. SOL_SOCKET indicates the Winsock-2 API defines the option.

optionname
The socket option for which the value is to be set.

optionvalue
The buffer in which the value for the requested option is supplied.

optionlength
The length of the value stored in *optionvalue*.

Option_name	Type	Description
SO_DONTLINGER	Boolean	Don't block closesocket() waiting for unsent data to be sent. Disconnect will be graceful. Setting this option is equivalent to setting SO_LINGER with *l_onoff* set to zero.
SO_LINGER	struct linger (*l_onoff*, *l_linger*)	Block closesocket() if unsent data is present. *l_onoff* is a boolean - 0 is don't linger, non-0 is linger *l_linger* is the linger timeout in seconds. Zero: abortive disconnect Non-zero: graceful disconnect
SO_OOBINLINE	Boolean	Receive out-of-band data in the normal data stream.
SO_RCVBUF	Integer	Specify buffer size for receives.
SO_SNDBUF	Integer	Specify buffer size for sends.

2.7 getsockopt()

Getsockopt returns the current socket options.

getsockopt (

IN	***socket,***
IN	***level,***
IN	***optionname,***
OUT	***optionvalue,***
IN OUT	***optionlength*****)**

socket
A socket previously allocated by a call to socket() or accept().

level
Which level of the stack defined the option. SOL_SOCKET indicates the Winsock-2 API defines the option.

optionname
The socket option for which the value is to be retrieved.

optionvalue
The buffer in which the value for the requested option is to be returned.

optionlength
On entry, the length of the buffer *optionvalue*. On exit, the length of the value stored in *optionvalue*.

See setsockopt() for the list of valid options.

3 Data Transfer API

Transport clients use the Data Transfer API to transfer data between clients.

It provides entries to:

- send data;
- receive data;
- control the mode of the connection.

3.1 send()

Send data over an open connection.

send (
IN ***socket,***
IN ***buffer,***
IN ***length,***
IN ***flags)***

socket
A socket to a connection opened by a call to accept() or connect().

buffer
Buffer of data to be sent over the connection.

length
Length of the data in *buffer*.

flags
Control the behavior of send()

Flag	Description
MSG_OOB	*Buffer* contains out-of-band data.
MSG_EOM	*Buffer* is the end of a message. *This is a proposed 1394PWG extension to Winsock-2.*

3.2 recv()

Receive data over an open connection. **Recv()** completes when data is received or when the connection is terminated by the remote client.

***bytes_received* = recv (**
IN ***socket,***
OUT ***buffer,***
IN ***length,***
IN ***flags*)**

bytes_received
The number of bytes received. If the connection has been gracefully closed, the return value is 0. Otherwise, a value of SOCKET_ERROR is returned.

socket
A socket to a connection opened by a call to accept() or connect().

buffer
Buffer for data to be received over the connection.

length
The length of *buffer*.

flags
Specifies the way in which the call is made

Flag	Description
MSG_PEEK	Peek at the incoming data. The data is copied into the buffer but is not removed from the input queue.
MSG_OOB	Receive out-of-band data (only valid when SO_OOBINLINE is disabled).

For byte stream style socket (e.g., type SOCK_STREAM), as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (setsockopt() - socket option SO_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. A client can use the SIOCATMARK ioctlsocket() command to determine whether there is any unread OOB data.

For message-oriented sockets (e.g., type SOCK_DGRAM), data is extracted from the first datagram (message) from the destination address specified in the connect() call. If the datagram or message is larger than the buffer supplied, the buffer is filled with the first part of the datagram, and **recv()** generates the error WSAEMSGSIZE. The data (*or is it excess data? - the spec is unclear*) is retained by the transport layer until it is successfully read by calling **recv()** with a large enough buffer.

If no incoming data is available at the socket, the **recv()** call waits for data to arrive unless the socket is non-blocking. In this case a value of SOCKET_ERROR is returned with the error code set to WSAEWOULDBLOCK.

3.3 ioctlsocket()

Control the mode of the socket.

ioctlsocket (
IN ***socket,***
IN ***command,***
IN OUT ***argumentpointer*****)**

socket
A socket to a connection opened by a call to accept() or connect().

command
The command to perform on *socket*.

argumentpointer
Pointer to the parameter for *command.*

Command	Parameter	Description
SIOCATMARK	Pointer to boolean return value	• Returns TRUE if next data in the data stream is *not* out-of-band data. • Returns FALSE if next data in the data stream is out-of-band data.
SIOEOMATMARK	Pointer to boolean return value	• Returns TRUE if next data in the data stream is the end of a message. • Returns FALSE if next data in the data stream is not the end of a message, or if *socket* is not of type *SOCK_STREAM* or if *socket* has not been enabled for in-line reception of out-of-band-data. *This is a proposed 1394PWG extension to Winsock-2.*

4 Disconnect API

Transport clients use the Disconnect API to close connections.

It provides entries to:

- shutdown data transfer on sockets;
- release sockets.

4.1 shutdown()

Disable sends and/or receives on a socket

shutdown (
IN ***socket,***
IN ***direction*****)**

socket
A socket to a connection opened by a call to accept() or connect().

direction
The direction(s) to be disabled. Can be SD_SEND, SD_RECEIVE, or SD_BOTH.

To assure that all data is sent and received on a connected socket before it is closed, a client should use shutdown() to close the connection before calling closesocket().

shutdown() does not close the socket. Resources allocated to the socket will remain reserved until the socket is closed using closesocket().

4.2 closesocket()

Closesocket() releases a socket. If the socket has an open connection, the connection is aborted before the socket is released. Any pending calls are canceled.

closesocket (
IN ***socket*****)**

socket
A socket previously allocated by a call to socket() or accept().

The semantics of closesocket() are affected by the socket options SO_LINGER and SO_DONTLINGER (see setsockopt()) as follows:

Option	**l_linger**	**Type of close**	**Wait for close?**
SO_DONTLINGER	Don't care	Graceful	No
SO_LINGER	Zero	Abortive	No
SO_LINGER	Non-zero	Graceful	Yes

5 Using the API

5.1 Opening and closing connections

5.1.1 Registering a server with the API

The following subroutine registers a service with the transport.

```
SOCKET register_service (unsigned __int64 unit_address,
                         char *service_name)
{
SOCKET      registered_socket;      // server's socket

if ((registered_socket = socket(AF_1394PWG, SOCK_STREAM,
IPPROTO_1394PWG)) != INVALID_SOCKET)
      {
      // socket has been created
      struct name {
            unsigned __int64  device_address;
            char        service_name;
            } this_service;

      this_service.device_address = unit_address;
      strcpy(this_service.service_name, service_name);
      if (bind(registered_socket, this_service, sizeof this_service)
            == 0)
            {
            // service name has been bound to socket
            return (registered_socket);
            }
      else
            return (SOCKET_ERROR);
      }
else
      return (SOCKET_ERROR);
}

The following example registers a server named "PRINT" with the
transport layer.

if ((server_socket = register_service(1394PWG_PRINT_UNIT_ADDRESS,
"PRINT")) != SOCKET_ERROR)
      {
      // service has been registered
      if (listen(server_socket, SOMAXCONN) == 0)
            {
            // socket is prepared to accept connections
            }
      }
```

5.1.2 Opening a connection to a server

Client:
The following subroutine creates a socket and opens a connection to a service.

```
SOCKET connect_to_service (unsigned __int64 target_address, char
*service_name)
{
SOCKET      client_socket;    // client's socket

if ((client_socket = socket(AF_1394PWG, SOCK_STREAM, IPPROTO_1394PWG))
!= INVALID_SOCKET)
      {
      // socket has been created
      struct name {
            unsigned __int64  device_address;
            char        service_name;
            } target_service;

      target_service.device_address = target_address;
      strcpy(target_service.service_name, service_name)
      if (connect(client_socket, target_service, sizeof this_service)
      == 0)
            {
            // connection is open
            return (client_socket);
            }
      else
            {
            return (SOCKET_ERROR);
            }
      }
else
      return (SOCKET_ERROR);
}
```

The following example opens a connection to a server named "PRINT".

```
if ((client_socket = connect_to_service(target_address, "PRINT")) !=
SOCKET_ERROR)
      {
      // connection is open
      }
```

Server:
The following example accepts a connection to socket *s*.

```
if ((connected_socket = accept(s, NULL, 0)) != INVALID_SOCKET)
      {
      // connection opened to server on connected_socket
      }
```

5.1.3 Closing a connection gracefully

The following subroutine closes a connection gracefully.

```
BOOLEAN close_connection (SOCKET    s)
{
if (shutdown(s, SD_SEND) != SOCKET_ERROR)
      {
      int ibytes_received;

      do  // read all remaining data
            ibytes_received = receive_data(s);
      while ((ibytes_received != 0)
            && (ibytes_received != SOCKET_ERROR));

      if (ibytes_received == 0)
            // remote socket issued shutdown - SD_SEND
            // shutdown both directions
            if (shutdown(s, SD_BOTH) != SOCKET_ERROR)
                  {
                  // release socket
                  if (closesocket(s) != SOCKET_ERROR)
                        return(TRUE);
                  }
      }
return(FALSE);
}
```

5.1.4 Aborting a connection

The following subroutine closes a connection in an abortive, or "hard" manner.

```
BOOLEAN abort_connection (SOCKET    s)
{
struct linger {
      u_short l_onoff;
      u_short l_linger;
      } linger_options;

linger_options.l_onoff = 1;   // SO_LINGER
linger_options.l_linger = 0;  // abortive close

if (setsockopt(s, SOL_SOCKET, SO_LINGER, &linger_options, sizeof
linger_options) != SOCKET_ERROR)
      // release socket
      if (closesocket(s) != SOCKET_ERROR)
            return(TRUE);
return(FALSE);
}
```

5.2 Data transfer

5.2.1 Sending data

The following code sends synchronous data.

```
// send will block until the transport has queued the data for sending
if (send(s, buffer, length, 0) != SOCKET_ERROR)
      // data was sent
```

5.2.2 Receiving data

The following subroutine receives synchronous data and delivers it to deliver(), a routine that processes the data:

```
int receive_data (SOCKET s)
{
int   ibytes_received;
char  buffer[RECEIVE_BUFFER];

// recv will block until data is received or connection is closed
if ((ibytes_received = recv (s, buffer, sizeof buffer, 0)) !=
SOCKET_ERROR)
      {
      deliver(&buffer, ibytes_received);
      return(ibytes_received);
      }
return(SOCKET_ERROR);
}
```

5.2.3 Sending out-of-band data

Sending out-of-band data is the same as sending normal data, with the MSG_OOB flag set.

```
// send will block until the transport has queued the data for sending
if (send(s, buffer, length, MSG_OOB) != SOCKET_ERROR)
      // out-of-band data was sent
```

5.2.4 Receiving out-of-band data

Reading out-of-band data requires the use of the ioctlsocket() entry.

```
int receive_data (SOCKET s)
{
BOOLEAN bnormal_data;
int   ibytes_received;
char  buffer[RECEIVE_BUFFER];

if (ioctlsocket (s, SIOCATMARK, &bnormal_data) != SOCKET_ERROR)
      if (bnormal_data)
            // recv will block until data received or connection closed
            if ((ibytes_received = recv (s, buffer, sizeof
                  buffer, 0)) != SOCKET_ERROR)
                  {
                  deliver(&buffer, ibytes_received);
                  return(ibytes_received);
                  }
      else
            // recv will block until data received or connection closed
            if ((ibytes_received = recv (s, buffer, sizeof
                  buffer, MSG_OOB)) != SOCKET_ERROR)
                  {
                  deliver_oob(&buffer, ibytes_received);
                  return(ibytes_received);
                  }
return(SOCKET_ERROR);
}
```

5.2.5 Sending stream messages