Universal Serial Bus Communications Class Subclass Specifications for Socket Management Protocol

Revision 0.1

September 4, 2019

Revision History

Table 1: Revision History

Version	Date	Comments
1.0	TBD	Initial Release

Contributors

Daniel Berliner	Xaptum
David Bild	Xaptum

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The Socket Control Module (SCM) subclass is a protocol by which USB devices can efficiently manage and use Berkeley/POSIX-style sockets on the host.

The specification currently defines support for the INET (IPv4) and INET6 (IPv6) protocol families and the TCP and UDP protocols. Future versions may add support for additional families and protocols.

This protocol offers an alternative method for a device to communicate with remote servers. Existing approaches that operate at the L3 layer (IP/USB) or L2 layer (CDC ECM, EEM, NCM) have several disadvantages. In particular, they require the device to be assigned an IP address. Since many networks will assign only a single IP address to a host, this requires implementing some form of network address translation (NAT) on the host. Setting up the NAT requires userspace configuration in most operating systems, so the USB device is no longer truly plug and play.

1.2 Scope

This document specifies new device subclasses intended for use with Communication devices, based on the Universal Serial Bus Class Definitions for Communication Devices specification [USBCDC].

The intention of this specification is that all material presented here be upwards-compatible extensions of the [USBCDC] specification. New numeric codes are defined for subclass codes, protocol codes, management elements, and notification elements.

In some cases material from [USBCDC] is repeated for clarity. In such cases, [USBCDC] shall be treated as the controlling document.

In this specification, the word 'shall' or 'must' is used for mandatory requirements, the word 'should' is used to express recommendations and the word 'may' is used for options.

2 Overview

Socket Control Model (SCM) is a specification for efficent management of sockets on a host by its devices. SCM sends Application Layer traffic and internal socket management commands from the device to the host over USB. This module requires host implementaitons for each suported Network and Transport layer protocols, but other layers are beyond its scope.

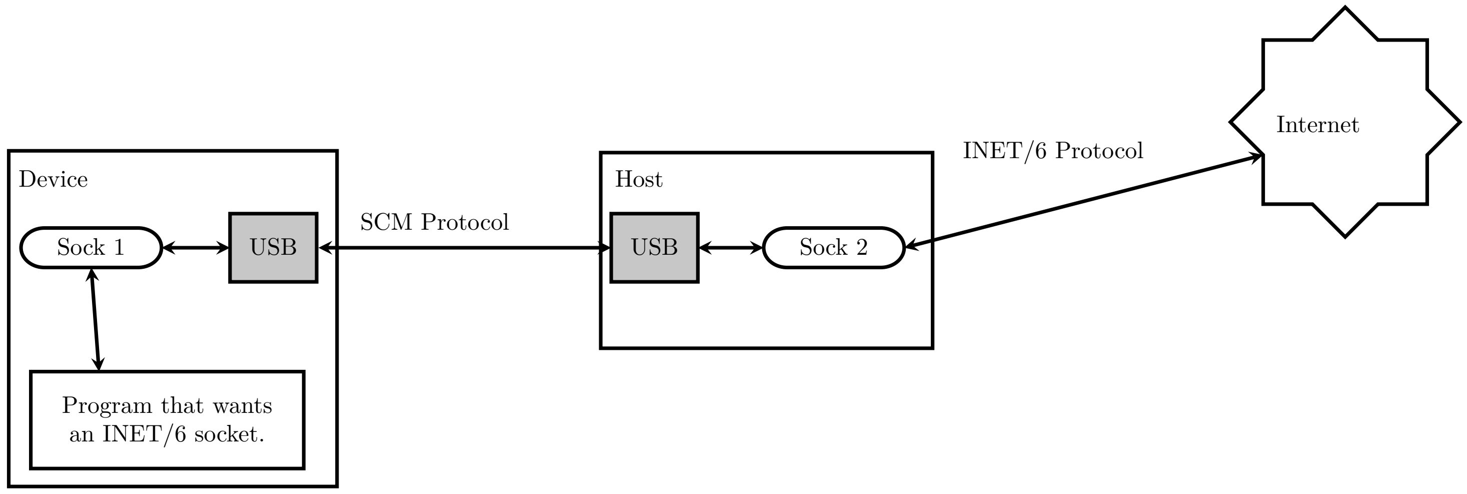
2.1 What is Socket Control Module (SCM)?

SCM allows a USB connected device to remotely create and control sockets on its host. A device may only control sockets it has created, and it shall expect exclusive control over this socket. Devices cannot control sockets created by other devices on the same host, and the host will not expose a devices socket for use by other applications.

Prior to SCM, connected devices would have to be assigned an IP address requiring the host to bridge the device or use NAT which also requires userspace configurations to the host. SCM allows a host to act on behalf of its device and provice it's network connection in a truly plug and play manner. This implicitly subjects the device to the same firewall rules and external network considerations without specific configuration.

The below figure illustrates a possible use case where a new socket type (Sock 1) is created on the device to interface with SCM. This allows userspace applications to interact with a standard protocol.

Figure 1: SCM Overview.



2.2 USB Endpoints

SCM's endpoint requirements consist soley of a bulk pair (In and Out). Socket commands are not transmitted over control endpoints becuase they are not controlling USB functions and they must arrive in order with writes.

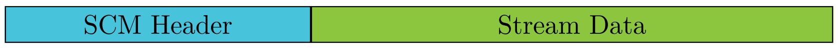
3 Socket Control Model (SCM)

The payload of the USB packet contains any combination of a single SCM packet, two or more SCM packets or a split SCM packet. SCM transfers can be split across USB packets but shall not be split across USB transfers.

3.1 SCM Transfer Format

The packet format defines an SCM packet. For information regarding USB packets refer to [USB2.0] specification. Details packets formats can be found in section 8.4 of the [USB2.0] specification.

An SCM packet consists of a 64-bit header and, depending on the command type, a varible length payload of arbitrary data. The length of this payload is indicated in the header. The payload will always immediately proceed the header.



3.2 SCM Transfer Types

The SCM packet header contains an opcode field (see section 1.3) to denote whether the transfer is an Immediate or Data type. Immediate types only contain the header while Data types contain a buffer of data immediately proceeding the packet of a length denotated in the header.

3.3 SCM Packets

All values in SCM packets are little endian. All IDs and integer values are unsigned unless otherwise denoted.

3.3.1 Common Fields

1. **Opcode**: 8-bit unsigned integer identifying what type of packet follows.

Table 2: Your first table.

Opcode	Name	Type	Purpose
0x00	OPEN	Cmd	Open a socket on the host
0x01	CONNECT	Data	Connect an open socket to a given address
0x02	CLOSE	Cmd	Disconnect and close a socket
0x03	WRITE	Data	Write data to a connected socket
0x04	ACK	Cmd	Acknowledge a command and indicate success
0x05	REPLY	Data	Similar to ACK but contains data.
0x06	IOCTL	Data	Tells the host to run ioctl on the sock.
0x07	SETOPT	Data	Tells the host to run setsockopt() on a sock.
0x08	GETOPT	Data	Tells the host to run getsockopt() on a sock.

2. **Message ID**: 8-bit unsigned integer provided to allow the receiver to identify the received message when replying. This is always genrated by the sender, and the receiver may only reply using the given ID once. Sending an ACK or REPLY to an unknown Message ID causes undefiend behavior.

3. **Sock ID**: 8-bit unsigned integer identifying which sock is being sent to. Sock IDs are always created by the device during an OPEN command.

3.3.2 SCM Command Packet

3.3.2.1 OPEN

The OPEN command is awlays initiated by the device to the host. The device will create a new message ID and new sock ID so the device can identify future operations on these objects.

0	8	16	24 31
0x00	Message ID	Sock ID	Reserved
Addr Family		Protocol	

Address Family is the ID used by the Linux kernel in `linux/socket.h`

Protocol is the ID used by the Linux kernel in `uapi/linux/in.h`

ACK will return as a 32-byte signed integer. On success ACK immediate will be 0, on failure the error code returned from the call.

3.3.2.2 CLOSE

When sent from device to host, disconnects (if connected) and closes a socket using the ID given during creation. If sent from host to device this will serve as a notification that the socket has been closed by the remote peer.

0	8	16	24 31
0x02	Message ID	Sock ID	Reserved
Exit Code (device to host only)			

On success ACK immediate will be 0, on failure the error code returned from the call (host to device only).

3.3.2.3 ACK

Upon completion of a message the receiver will send this back to acknowledge reciept and indicate whether the operation was a succes or a failure. Once USB has acknowledged reciept, the sender of an ACK will not wait for further confirmation that the recipient has received the message.

0	8	16	24 31
0x04	Message ID	Sock ID	Reserved
See ACK section on commands			

3.3.3 SCM Data Packet

Data packets contain arbitrary data immediately after the header of whatever length is contained in the headers length field. The data segment of specific commands may represent structures with endianness, all fields are individually converted to little endian before sending and converted back when received.

Data Length will always be an unsigned 32-bit integer.

3.3.3.1 CONNECT

Connect tells the host to connect a created socket to a given address. The address information passed will vary by protocol, the host and device should know which structs the other side will send and process those (these are typically the same on both ends).

Table 3: Values for Family.

ID	Name
0x01	IP
0x02	IP6

Table 4: Values for Protocol.

ID	Name
0x01	TCP
0x02	UDP

IP Connect Packet

0	8	16	24	32
0x01	Message ID	Sock ID	Reserved	
0x10				
0x00	Protocol	Port in Network Byte Order[1]		
IP Address in Network Byte Order				

IP6 Connect Packet

0	8	16	24	32
0x01	Message ID	Sock ID	Reserved	
0x24				
0x01	Protocol	Port in Network Byte Order		
flowinfo				
Scope ID				
IP6 Addr In Network Byte Order				
...IP6 Addr In Network Byte Order...				
...IP6 Addr In Network Byte Order...				
...IP6 Addr In Network Byte Order				

[1]POSIX.1-2017 expects the address and port to be in network byte order regardless of the host byte order.

3.3.3.2 WRITE

Sends stream data over a connected socket. This command can be sent by either side.

0	8	16	24 31
0x03	Message ID	Sock ID	Reserved
Payload Length in bytes			
Payload data...			

ACK will return as a 32-byte signed integer. On error the return code (¡0) will be returned. Zero will be returned on success and positive codes will be returned when the transfer was a success but the receiver needs to tell the sender to change sending behavior (TODO: IMPLEMENT POSITIVE CODE BEHAVIOR)

3.3.3.3 REPLY

Similar to ACK but contains a data field for internal processing.

0	8	16	24 31
0x05	Message ID	Sock ID	Reserved
Payload length			
Payload data...			

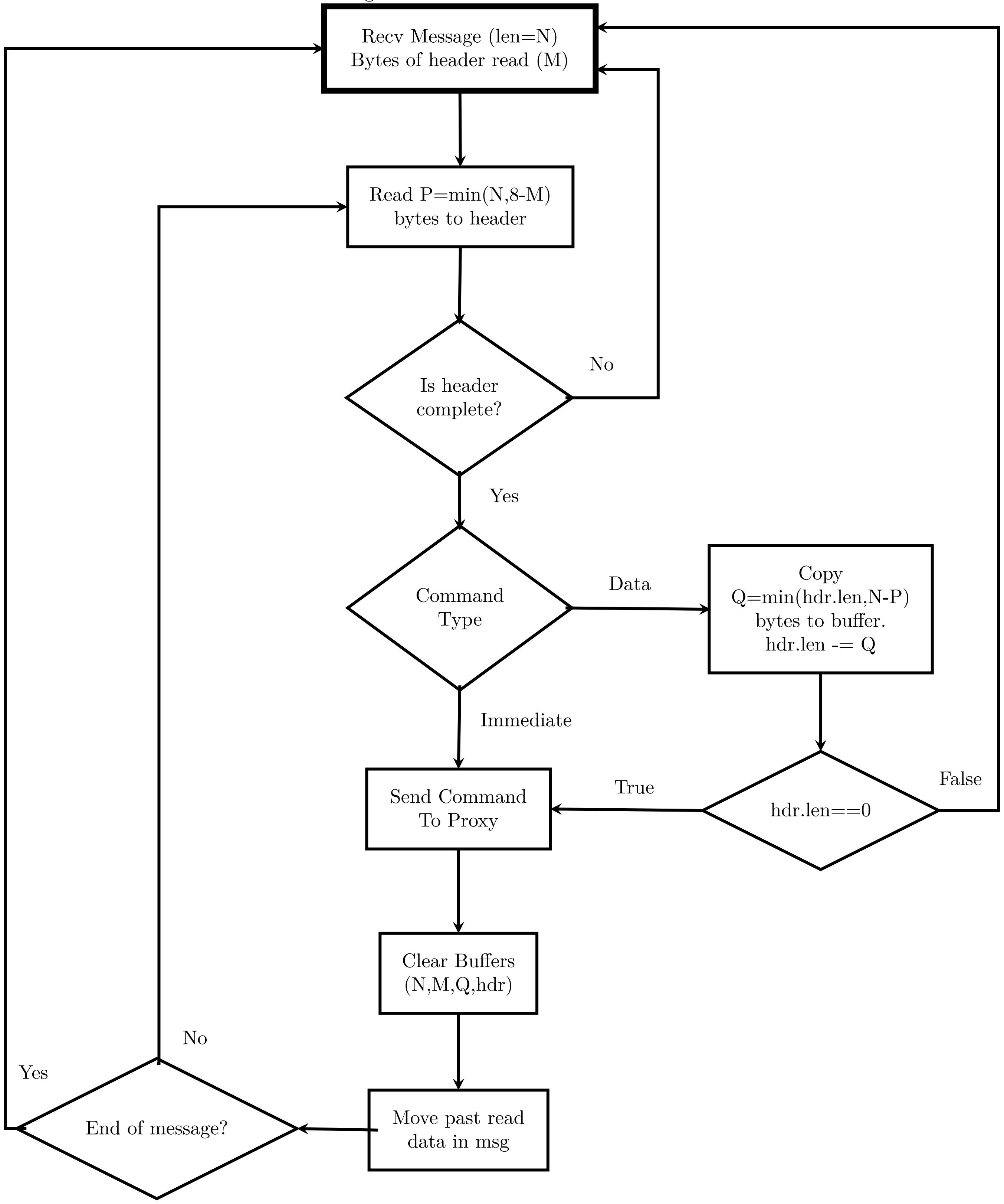
4 Packet Processing

Fig A describes the process for the receiver on both ends to assemble and transmit packets from USB to their respective proxies. The Send Command To Proxy procedure can be assumed to be nonblocking, after which the data passed in can be safely freed.

4.1 Processing Flow

4.1.1 Receiving SCM Packets From USB

Figure 2: SCM Packet receive Flow.



Recv Message (len=N)
Bytes of header read (M)
Read P=min(N,8-M) bytes to header
Is header complete?
No
Yes
Command Type
Data
Copy Q=min(hdr.len,N-P) bytes to buffer. hdr.len -= Q
Immediate
Send Command To Proxy
True
hdr.len==0
False
Clear Buffers (N,M,Q,hdr)
No
Yes
End of message?
Move past read data in msg