CoolTerm Remote Control Socket

Protocol Specification v0.9.2

Roger Meier, April 2020

Table of Contents

List of Figures

List of Tables

1. Introduction

This document specifies a protocol, based on TCP/IP, which allows actions, normally performed via the CoolTerm GUI, to be automated by a separate piece of software (e.g. scripting software). A listening TCP socket embedded in CoolTerm (Remote Control Socket), which is enabled via the CoolTerm GUI, can accept connections from the same computer on which CoolTerm is running as well as other computers that can make a TCP/IP connection to the computer on which CoolTerm is running. Another application that is connected to the Remote Control Socket can send commands to initiate actions normally performed via the GUI (e.g. open/closing the serial port, reading/writing data, etc.).

2. CoolTerm Remote Control Socket Protocol

2.1. Overview

The CoolTerm Remote Control Socket Protocol is based on TCP/IP and is therefore a Client/Server type protocol. The CoolTerm application acts as the server while an external application (e.g. scripting application) acts as a client. Connections and subsequent data communication are initiated only by the client. I.e. the client can connect to and disconnect from a server socket, and only the client can initiate communication with the server. The server cannot send any unsolicited data.

2.2. Server

The CoolTerm application has an embedded Remote Control Socket that is configured as server. The socket is normally disabled, but it can be enabled via the CoolTerm GUI. If enabled, the socket listens on a specified port for incoming connections. Once connected, the server waits for incoming packets. The server always responds to packets from the client to acknowledge them and to return data asked for by the client. The server does not send any unsolicited data.

The specifications for the Remote Control Socket configured as server are as follows:

- Default Port: 51413
- Normally disabled. Can be enabled via CoolTerm GUI.
- Always acknowledges receipt of a valid Remote Packet with another Remote Packet, i.e. ACK_SUCCESS, together with data requested by the client (if necessary).
- Always acknowledges receipt of invalid Remote Packets with the appropriate ACK code, i.e. ACK_BAD_OPCODE, ACK_BAD_ARGUMENT, etc.
- Always acknowledges receipt of incomplete Remote Packets with the appropriate ACK code after a specified timeout, i.e. ACK_TIMEOUT
- Default timeout for incomplete packets: 1 second.

2.3. Client

The client is an application that connects to the server on a specified port, using an embedded Remote Control Socket configured as client. Once a connection with the server is established, it is the responsibility of the client to drive the communication. The server will not send any data without a request from the client. The server will acknowledge any Remote Packet sent by the client with a response. If data is requested by the client, the server will attach the requested data to the response. The client always expects a response from the server for any sent packet. If no response is received within a specified timeout, it is the responsibility of the client application to either retry the communication or alert the user.

2.4. Remote Packet

2.4.1. Remote Packet Format

The Remote Packet format is depicted in Figure 1.

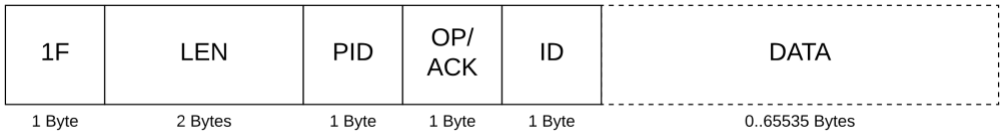

Figure 1: Remote Packet Format.

A Remote Packet is comprised of the following components:

- **1F:** This preamble is present at the beginning of all Remote Packets. This allows the Remote Control Socket to recognize the beginning of a new packet. As the name suggests, the value is *0x1F*.
- **LEN:** This is the length field of the packet. Its value is the length of the DATA field. LEN is UInt16, and the byte-order is little endian.
- **PID:** This is the Packet ID. It is the client's responsibility to pick a new Packet ID for every new packet. The server will respond to received packets by using the received packet ID in its response. This allows the client to associated sent packets with corresponding response (ACK) packets from the server.
- **OP/ACK:** This field is used for the OP (Operator) code for packets sent from the client to the server, and for the ACK (Acknowledge) code for packets sent from the server to the client.
- **ID:** This is the terminal ID to which the packet is to be directed. Each CoolTerm terminal window has its own, unique, terminal ID. This allows OP packets to be addressed to specific terminal windows. While not all OP codes are address to specific terminal windows, the ID byte needs to be present in the packet regardless (the actual value will be ignored by the server). Responses from the server will always be addressed to *0xFF*

- **DATA:** If data is to be sent, it done is via the DATA field of the packet. The DATA field can contain 0 to 65535 bytes. The DATA field is to be formatted as a character string.

Packets sent from the Client to the Server always contain an OP Code. The DATA field is only populated if required by the OP Code.
Packets sent from the Server to the Client always contain an ACK Code. Data requested by the Client will be sent via the DATA field.

2.4.2. Examples

The following examples illustrate possible communications between Client and Server. Refer to 2.4.3 and 2.4.4 for details on OP and ACK codes, respectively.

Example 1: The following example, the Client sends an OP_PING command to the server, and the server responds with and ACK_SUCCESS code. The packet bytes are shown in hexadecimal format:

Packet sent by Client: **`1F 00 00 DF 00 00`**

- LEN: 0x0000 (0 Bytes)
- PID: 0xDF
- OP: 0x00 (1: ping)
- ID: 0x00

Response sent by Server: **`1F 00 00 DF FF FF`**

- LEN: 0x0000 (0 Byte)
- PID: 0xDF
- ACK: 0xFF (255: success)
- ID: 0xFF

Example 2: In this example, the client requests the name of the window with index 3 from CoolTerm:

Packet sent by Client: **`1F 01 00 E8 1A 00 33`**

- LEN: 0x0000 (0 Bytes)
- PID: 0xE8
- OP: 0x1A (26: GetWindowName)
- ID: 0x00
- DATA: 0x33 (“3”)

Response sent by Server: **`1F 0A 00 E8 FF FF 43 6F 6F 6C 54 65 72 6D 5F 30`**

- LEN: 0x000A (10 Bytes)
- PID: 0xE8
- ACK: 0xFF (255: success)

- ID: 0xFF
- DATA: "CoolTerm_0")

2.4.3. Remote Packet OP Codes

The Remote Protocol consists of, but is not limited to, the OP Codes listed in Table 1 below.

System Commands			
Description	**OP**	**Data**	**Return Data**
OP_PING *Causes the Server to return an ACK_SUCCESS packet if a sound processor is online and ACK_OFFLINE if no sound processor is currently online.*	0	-	-
OP_LAST_SOCKET_ERROR *Returns the error code for the last socket error. Returns 0 for no error.*	1	-	LastSocketError as String
Window/App Commands			
Description	**OP**	**Data**	**Return Data**
OP_NEW_WINDOW *Opens a new CoolTerm window. Returns the ID of the new window.*	20	-	ID as String
OP_LOAD_SETTING *Instructs CoolTerm to load the connection settings specified by the FilePath. Returns the ID of the new window if loading was successful, or -1 if it was not.*	21	FilePath as String *FilePath can be either absolute or relative to the location of the CoolTerm executable.*	ID as String
OP_SAVE_SETTING *Instructs CoolTerm to save the settings of the terminal window specified by WindowName at the path specified by FilePath*	22	FilePath as String *FilePath can be either absolute or relative to the location of the CoolTerm executable.*	Success as String *"True": Success* *"False": No Success*
OP_GET_WINDOW_COUNT *Returns the number of open terminal windows.*	23	-	WindowCount as String
OP_GET_WINDOW_ID *Returns the ID of the window with the specified Index, or -1 if the index is invalid.*	24	Index as String *[0..WindowCount-1]*	ID as String
OP_GET_WINDOW_ID_FROM_NAME *Returns the ID of the window with the specified name, or -1 if the window doesn't exist.*	25	WindowName as string	ID as String
OP_GET_WINDOW_NAME *Returns the name of the terminal window with the specified index, or an empty String if the index is invalid.*	26	Index as String *[0..WindowCount-1]*	Name as String
OP_INDEX_OF_WINDOW_ID *Returns the Index of the window with the specified ID.*	27	-	Index as String
OP_CLOSE_WINDOW *Closes the window with the specified ID.*	28	-	-

OP_QUIT *Quits CoolTerm.*	29	-	-
OP_VERSION *Returns the CoolTerm version.*	30	-	CoolTermVersion as String
OP_SHOW_WINDOW *Brings the window with the specified ID to the front.*	31	-	
OP_PRINT Prints the current contents of the window with the specified ID.	32	-	Success as String *"True": Success* *"False": No Success*
Serial Port Commands			
Description	**OP**	**Data**	**Return Data**
OP_CONNECT *Opens the serial port. Returns True on success.*	40	-	Success as String *"True": Success* *"False": No Success*
OP_DISCONNECT *Closes the serial port.*	41	-	-
OP_IS_CONNECTED *Returns True if the serial port is open.*	42	-	Success as String *"True": Success* *"False": No Success*
OP_LAST_ERROR *Returns the last serial port error code.*	43	-	ErrorCode as String
Data Exchange Commands			
Description	**OP**	**Data**	**Return Data**
OP_WRITE *Writes data to the serial port.*	50	Data as String	-
OP_WRITE_LINE *Writes data terminated by the "Enter Key Emulation" character specified in the connection settings to the serial port.*	51	Data as String	-
OP_WRITE_HEX *Writes Hex formatted data to the serial port. This is useful when transmitting binary data that can't be expressed with a regular character string.*	52	HexData as String	-
OP_BYTES_LEFT_TO_SEND *Returns the number of bytes left in the transmit buffer awaiting transmission.*	53	-	NumBytes as String
OP_POLL *Polls the serial port. This causes all data currently available in the serial port receive buffer to be transferred to CoolTerm's receive buffer immediately. It is recommended to call this method before calling OP_READ, OP_READ_HEX, OP_READ_ALL, OP_LOOK_AHEAD, OP_LOOKAHEAD_HEX, and OP_BYTES_AVAILABLE.*	54	-	-
OP_READ *Reads and removes the specified number of characters from the receive buffer.*	55	NumBytes as String	Data as String
OP_READ_ALL *Reads and removes all characters from the receive buffer.*	56	-	Data as String

OP_READ_HEX *Reads and removes the specified number of characters from the receive buffer. Returns the read data in Hex format.*	57	-	HexData as String
OP_READ_ALL_HEX *Reads and removes all characters from the receive buffer. Returns the read data in Hex format.*	58	-	HexData as String
OP_BYTES_AVAILABLE *Returns the number of characters currently available in the receive buffer.*	59	-	NumberOfBytes as string
OP_LOOK_AHEAD *Returns the contents of the receive buffer without removing any data.*	60	-	Data as String
OP_LOOK_AHEAD_HEX *Returns the contents of the receive buffer in Hex format without removing any data.*	61	-	HexData as String
OP_CLEAR_BUFFER *Clears receive buffer.*	62	-	-
Serial Commands			
Description	**OP**	**Data**	**Return Data**
OP_SEND_BREAK *Sends a break signal.*	70	-	-
OP_FLUSH_PORT *Flushes the Serial Port Buffers.*	71	-	-
OP_RESET_PORT *Resets the Serial Port.*	72	-	-
OP_GET_DTR *Returns the state of the DTR status line.*	73	-	State as String *"True": active* *"False: inactive*
OP_SET_DTR *Sets the state of the DTR status line.*	74	State as String *"True": active* *"False: inactive*	-
OP_GET_RTS *Returns the state of the RTS status line.*	75	-	State as String *"True": active* *"False: inactive*
OP_SET_RTS *Sets the state of the RTS status line.*	76	State as String *"True": active* *"False: inactive*	-
OP_GET_CTS *Returns the state of the CTS status line.*	77	-	State as String *"True": active* *"False: inactive*
OP_GET_DSR *Returns the state of the DSR status line.*	78	-	State as String *"True": active* *"False: inactive*
OP_GET_DCD *Returns the state of the DCD status line.*	79	-	State as String *"True": active* *"False: inactive*
OP_GET_RI *Returns the state of the RI status line.*	80	-	State as String *"True": active* *"False: inactive*

Text Data Exchange Commands			
Description	**OP**	**Data**	**Return Data**
OP_SEND_TEXTFILE *Sends the text file with the specified FilePath..*	90	FilePath as String *FilePath can be either absolute or relative to the location of the CoolTerm executable.*	Success as String *"True": Success* *"False": No Success*
OP_CAPTURE_START *Starts capture of data to the text file at the specified FilePath*	91	FilePath as String *FilePath can be either absolute or relative to the location of the CoolTerm executable.*	Success as String *"True": Success* *"False": No Success*
OP_CAPTURE_PAUSE *Pauses a Capture currently in progress.*	92	-	-
OP_CAPTURE_RESUME *Resumes a previously paused Capture.*	93	-	-
OP_CAPTURE_STOP *Stops a capture currently in progress and closes the file.*	94	-	-
Connection Setting Commands			
Description	**OP**	**Data**	**Return Data**
OP_RESCAN_SERIALPORTS *Rescans the system for available serial ports.*	100	-	-
OP_GET_SERIALPORT_COUNT *Returns the number of available serial ports.*	101	-	SerialPortCount as String
OP_SERIALPORT_NAME *Returns the name of the Serial Port with the specified index, or an empty String if the index is invalid.*	102	SerialPortIndex as String *[0..SerialPortCount-1]*	SerialPortName as String
OP_GET_CURRENT_SERIALPORT *Returns the index of the currently selected Serial Port.*	103	-	SerialPortIndex as String
OP_SET_CURRENT_SERIALPORT *Selects the serial port with the specified index. This can only be done while the port is closed. Returns True on success.*	104	SerialPortIndex as String *[0..SerialPortCount-1]*	Success as String *"True": Success* *"False": No Success*
OP_GET_PARAMETER *Returns the value of parameter specified by ParameterName.* *To obtain a list of all available Parameter names, use OP_GET_ALL_PARAMETERS.*	110	ParameterName as String	Value as String
OP_SET_PARAMETER *Returns the value of the parameter specified by ParameterName. ParameterName and Value need to be separated by the NUL (ASCII 0) character. Returns True on success.* *To obtain a list of all available Parameter names, use OP_GET_ALL_PARAMETERS.*	111	ParameterName + NUL + Value as String	Success as String *"True": Success* *"False": No Success*
OP_GET_ALL_PARAMETERS *Returns a list of all parameter names their values (one per line).*	112	-	ParameterList as String

Table 1: Remote Packet OP Codes.

It is the responsibility of the Server (i.e. CoolTerm) to execute the proper operations upon receipt of one of these OP packets. It is also the Server's responsibility to verify the validity of received packets and respond to the client accordingly using ACK Codes.

2.4.4. Remote Packet ACK Codes

The Remote Protocol consists of, but is not limited to, the ACK Codes listed in Table 2 below.

Description	ACK
ACK_SUCCESS	255
ACK_BAD_OPCODE	254
ACK_BAD_ARGUMENT	253
ACK_TIMEOUT	252
ACK_OFFLINE	251

Table 2: Remote Packet ACK Codes.

- **ACK_SUCCESS:** This code is used by the Server to indicate to the Client that the packed was successfully received and to return data requested by the Client in its DATA field.
- **ACK_BAD_OPCODE:** This code is sent by the server if the OP code in the received packet is invalid
- **ACK_BAD_ARGUMENT:** This code is sent by the server if the argument contains invalid values (outside the valid number range) or has an invalid format (e.g. Byte instead of UInt16). The server also returns this code the ID field in the received OP packet is invalid.
- **ACK_TIMEOUT:** This code is used by the server to indicate to the client that it has not received a complete package within a specified time frame (default: 1 second).
- **ACK_OFFLINE:** This code is returned by the server to indicate to the client that no sound processor is online.

Upon receipt of an ACK code that indicates an error, it is the responsibility of the Client software to either retry the communication or to alert the user.