Close Talk Conference System, Remote Audio Administration

Version 1 - 2016-03-14

NOTE! In order to use these audio administration commands, the Central Unit requires firmware version 2.15 or higher! Use the SERCOMMCOMMAND__GET_UNIT_INFO command to programmatically verify the installed firmware version before using these commands.

Message formatting

The data links, RS-232 and wireless infrared, uses a simple binary (to conserve bandwidth) message format. The protocol is based on Pull-messaging, i.e. all events and data messages must be requested, no unit transmits unless requested.

The message format is constructed as follows:

ASCII_SOH, Counter, ID 3, ID 2, ID 1, … Data …, Control

where each character is a single 8 bit byte. The message parts are:

ASCII_SOH ASCII Start Of Header, hex 0x01.

Counter The total number of bytes in the message including *SOH*, *ID*, *Data* and *Control*.

ID3,2 and 1 For *Query* and *Control* commands, the target unit ID number as unsigned 24 bits were ID 3 is bits 23-16, ID 2 bits 15-8 and ID 1 bits 7-0. When sending over the infrared data link the ID should be set to the targeted Delegate Unit or to 0 for broadcast (all delegate units). For *Response* messages the responding unit inserts its own ID number in the response message. When communicating with the Central Unit over the RS-232 port the ID number is ignored (set to 0). The Delegate Unit only responds to personal (its own ID number) or broadcast messages (ID set to 0).

Data Message specific data as described for each message type. Number of data is *Counter* – 6. The first byte of the *Data* part is always the command specifier. number. The *Data* part can be a maximum of 32 bytes long.

Control The *Control* byte is a checksum for integrity control and is constructed by signed 8 bit addition with ignored overflow of all message characters except *Control*. The resulting sum is then 2's-complemented and subtracted by 1 for the final *Control* byte. A message is only processed if the checksum is correct.

A message can be one of three types:

- *Query*: A *Query* message results in a *Response* message sent back from the receiver where the *Response* message contains the requested data
- *Command* : A *Command* message results in a SERCOMMCOMMAND__POSITIVE_ ACKNOWLEDGE *Response* message if the command succeeded or a SERCOMMCOMMAND__NEGATIVE_ACKNOWLEDGE if the command fails
- *Response*: Either a POSITIVE/NEGATIVE response to a *Command* or the requested data for a *Query* message. ***Note*** that the *Data* part of a *Response* message always has the first byte set to the *Message specifier number*, the actual response data (if any) begins at *Data* part byte 2

The rest of the document describes each available message. The *Message* line is the message name followed by the message specifier number in brackets. The *Data* line shows the message *Data* part format. The Description part describes the message action and possible *Response* data.

Use the *Central Unit User Manual* as a reference for parameter limits and the different parts of the audio system. ***Note*** that the Central Unit front panel is still operational during the use of these commands, simultaneous use may cause underside operation.

The Central Unit RS-232 port settings are fixed to 9600 Baud, hardware flow control, 8 data bits, No parity, 1 stop bit. The serial port pin-out is adapted to commercially available null-modem cables, see the Central Unit User Manual for connection diagram.

Command List

Message: SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE (2)
Data: **2**

Description:

Response message, indicates that the received message was executed correctly.

Message: SERCOMMCOMMAND__NEGATIVE_ACKNOWLEDGE (3)
Data: **3**

Description:

Response message, indicates that the received message execution failed.

Message: SERCOMMCOMMAND__GET_UNIT_INFO (5)
Data: **5**

Description:

Responds with information about the Central Unit. The response formatting is *CU prog. vers. 1, CU prog. vers. 2, Unit Type, Unit Variant, DU prog. vers. 1, DU prog. vers. 2, Serial number 3, Serial number 2, Serial number 1* where:

CU prog. vers. 1 + *CU prog. vers. 2* is the Central Unit's fimware version as unsigned 16 bits where *CU prog. vers. 1* is bits 15 - 8 and *CU prog. vers. 2* is bits 7 - 0. The program version is sent *Version*100*, for example firmware version 2.15 is sent as 215.

Unit Type indicates which unit type that is responding, always 1 for the Central Unit.

Unit Variant is not used, backward compatibility only.

DU prog. vers. 1 + *DU prog. vers. 2* is the Delegate Unit's firmware version stored in the Central Unit as as unsigned 16 bits where *DU prog. vers. 1* is bits 15 - 8 and *DU prog. vers. 2* is bits 7 - 0. The program version is sent *Version*100*, for example firmware version 2.08 is sent as 208.

Serial number 1-3 is the Central Unit's serial number as a 24 bit unsigned number where *Serial number 3* is bits 23-16, *Serial number 2* is bits 15-8 and *Serial number 2* is bits 7-0.

Message: SERCOMMCOMMAND__SET_SPEAKER_SOUND_VOLUME (11)
Data: **11**, Base level, Max level, Store

Description:

Sets Delegate Unit speaker audio *Base level* (volume knob fully counter-clockwise) and *Max level* (volume knob fully clockwise). Setting both levels to the same number fixes the speaker audio level (delegate volume knob is disabled). Use the SERCOMMCOMMAND __GET_SPEAKER_SOUND_VOLUME command to learn the level limits. *Base level* must be equal to or lower than *Max Level*.

Store is Boolean, set to 0 to make the change temporary or 1 to make the setting be stored permanently in the Central Unit. For remote operations, 0 is most common. Returns SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE if successful.

Message: SERCOMMCOMMAND__SET_HEADPHONES_SOUND_VOLUME (12)
Data: **12**, Base level, Max level, Store

Description:

Sets Delegate Unit headphones audio *Base level* (volume knob fully counter-clockwise) and *Max level* (volume knob fully clockwise). Setting both levels to the same number fixes the headphones audio level (delegate volume knob is disabled). Use the SERCOMMCOMMAND __GET_ HEADPHONES _SOUND_VOLUME command to learn the level limits. *Base level* must be equal to or lower than *Max Level*.

Store is Boolean, set to 0 to make the change temporary or 1 to make the setting be stored permanently in the Central Unit. For remote operations, 0 is most common. Returns SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE if successful.

Message: SERCOMMCOMMAND__GET_GLOBAL_EQ (22)
Data: **22**

Description:

Queries the Central Unit global audio EQ levels. Returns *Low*, *Low MAX*, *Low MIN*, *High*, *High MAX*, *High MIN* where *Low* is the low register (base) level and *High* is the high register (treble) level. The *xxx MIN* and *xxx MAX* data is the minimum&maximum levels available for the EQ.

Message: SERCOMMCOMMAND__SET_GLOBAL_EQ (23)
Data: **23**, Low, High

Description:

Sets the Central Unit global audio EQ levels. *Low* is the low register (base) level and *High* is the high register (treble) level. Use the SERCOMMCOMMAND__GET_GLOBAL_EQ command to learn the maximum and minimum levels available. If the command is successful SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE is returned, otherwise SERCOMM COMMAND__NEGATIVE_ACKNOWLEDGE.

Message: SERCOMMCOMMAND__GET_GLOBAL_VOLUME (24)
Data: **24**

Description:

Queries the Central Unit global audio level (see CU manual). Returns *Level*, *Level MAX*, *Level MIN*. *Level* is the current level, *xxx MIN* and *xxx MAX* indicates the max and min levels the Central Unit can accept.

Message: SERCOMMCOMMAND__SET_GLOBAL_VOLUME (25)
Data: **25**, Level

Description:

Sets the Central Unit global audio level to *Level*. Use the SERCOMMCOMMAND__ GET_GLOBAL_VOLUME command to learn the maximum and minimum levels available. If the command is successful SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE is returned, otherwise SERCOMMCOMMAND__NEGATIVE_ACKNOWLEDGE.

NOTE! *Do not* use this command to specify the Delegate Unit speaker audio levels, only set this to the default level (On) or to 0 (Off)! See Central Unit manual for more information.

Message: SERCOMMCOMMAND__GET_LINE_IN_VOLUME (28)
Data: **28**

Description:

Queries the Central Unit *Line In* audio level. Returns *Level*, *Level MAX*, *Level MIN*. *Level* is the current level, *xxx MIN* and *xxx MAX* indicates the max and min levels the Central Unit can accept.

Message: SERCOMMCOMMAND__SET_LINE_IN_VOLUME (29)
Data: **29**, Level

Description:

Sets the Central Unit *Line In* audio level to *Level*. Use the SERCOMMCOMMAND__ GET_LINE_IN_VOLUME command to learn the maximum and minimum levels available. If the command is successful SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE is returned, otherwise SERCOMMCOMMAND__NEGATIVE_ACKNOWLEDGE.

Message: SERCOMMCOMMAND__GET_TELE_IN_VOLUME (32)
Data: **32**

Description:

Queries the Central Unit *Tele In* audio level. Returns *Level*, *Level MAX*, *Level MIN*. *Level* is the current level, *xxx MIN* and *xxx MAX* indicates the max and min levels the Central Unit can accept.

Message: SERCOMMCOMMAND__SET_TELE_IN_VOLUME (33)
Data: **33**, Level

Description:

Sets the Central Unit *Tele In* audio level to *Level*. Use the SERCOMMCOMMAND__ GET_TELE_IN_VOLUME command to learn the maximum and minimum levels available. If the command is successful SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE is returned, otherwise SERCOMMCOMMAND__NEGATIVE_ACKNOWLEDGE.

Message: SERCOMMCOMMAND__GET_LINE_IN_TO_TELE_OUT_VOLUME (34)
Data: **34**

Description:

Queries the Central Unit *Line In to Tele Out* cross-coupling audio level. Returns *Level*, *Level MAX*, *Level MIN*. *Level* is the current level, *xxx MIN* and *xxx MAX* indicates the max and min levels the Central Unit can accept.

Message: SERCOMMCOMMAND__SET_LINE_IN_TO_TELE_OUT_VOLUME (35)
Data: **35**, Level

Description:

Sets the Central Unit *Line In To Tele Out cross* coupling audio level to *Level*. Use the SERCOMMCOMMAND__GET_LINE_IN_TO_TELE_OUT_VOLUME command to learn the maximum and minimum levels available. If the command is successful SERCOMM COMMAND__POSITIVE_ACKNOWLEDGE is returned, otherwise SERCOMMCOMMAND__ NEGATIVE_ACKNOWLEDGE.

Message: SERCOMMCOMMAND__GET_SPEAKER_SOUND_VOLUME (43)
Data: **43**

Description:

Queries the Delegate Unit speaker levels. Returns *Base Level*, *Max Level*, *Roof Level*, *Floor Level*. *Base level* is the current level with the Delegate Unit volume knob turned fully counter-clockwise and *Max level* is the current level with the volume knob fully clockwise.

Roof Level is the maximum level accepted and *Floor Level* is the minimum level accepted for the *Base & Max Level* parameters in the SERCOMMCOMMAND__ SET_SPEAKER_SOUND_VOLUME command.

Message: SERCOMMCOMMAND__GET_HEADPHONES_SOUND_VOLUME (44)
Data: **44**

Description:

Queries the Delegate Unit headphones levels. Returns *Base Level*, *Max Level*, *Roof Level*, *Floor Level*. *Base level* is the current level with the Delegate Unit volume knob turned fully counter-clockwise and *Max level* is the current level with the volume knob fully clockwise.

Roof Level is the maximum level accepted and *Floor Level* is the minimum level accepted for the *Base & Max Level* parameters in the SERCOMMCOMMAND__ SET_HEADPHONES_SOUND_VOLUME command.

Message: SERCOMMCOMMAND__ GET_LINE_OUT_VOLUME (77)
Data: **77**

Description:

Queries the Central Unit *Line Out* audio level. Returns *Level*, *Level MAX*, *Level MIN*. *Level* is the current level, *xxx MIN* and *xxx MAX* indicates the max and min levels the Central Unit can accept.

Message: SERCOMMCOMMAND__ SET_LINE_OUT_VOLUME (78)
Data: **78**, Level

Description:

Sets the Central Unit *Line Out* audio level to *Level*. Use the SERCOMMCOMMAND__ GET_LINE_OUT_VOLUME command to learn the maximum and minimum levels available. If the command is successful SERCOMMCOMMAND__POSITIVE_ACKNOWLEDGE is returned, otherwise SERCOMMCOMMAND__NEGATIVE_ACKNOWLEDGE.

Message: SERCOMMCOMMAND__ GET_AUDIO_CHANNEL_USAGE (82)
Data: **82**

Description:

Queries the activity of the Central Unit wireless audio channels. The response message is *A1-3*, *A1-2*, *A1-1*, *A2-3*, *A2-2*, *A2-1*, *A3-3*, *A3-2*, *A3-1* where Ax is the wireless audio channel number 1 to 3 and Ax-1 to 3 is an unsigned 24 bit number that forms the ID number of the Delegate Unit currently active on that channel. Ax-1 is bits 23 - 16, Ax-2 is bits 15 - 8 and Ax-1 is bits 7 - 0. If the channel is free (no activity), ID number 0 is returned. Poll the Central Unit at regular intervals, for example once per second. See the *Central Unit User Manual* for additional information.

Code Examples

These code examples are in C. They are not directly compilable and should be considered as a reference or example to base the actual code on.

Message encoder:

```
/********************************************************************************************************/
void Construct_Message_And_Start_Transmitter(unsigned long Serial, unsigned char *Data,
unsigned char DataCount)
{
if ((DataTransmitterBuffer__Buffer_Contents_Size + Count + 6) < TX_BUFFER_SIZE)
   {
   /* Message fits, start constructing */
   signed char Hold;
   signed char CS = ASCII_SOH;
   unsigned char Data_Counter;

   /* Add SOH character to transmitter */
   Add_Character_To_Transmitter_Buffer(ASCII_SOH);



   /* Add Message byte counter to checksum and transmitter */
   Hold = Count + 6;
   CS += Hold;
   Add_Character_To_Transmitter_Buffer(Hold);



..../* Add the three ID bytes characters to checksum and transmitter */
   Hold = *((signed char *)&Serial + 1);
   CS += Hold;
   Add_Character_To_Transmitter_Buffer(Hold);

   Hold = *((signed char *)&Serial + 2);
   CS += Hold;
   Add_Character_To_Transmitter_Buffer(Hold);

   Hold = *((signed char *)&Serial + 3);
   CS += Hold;
   Add_Character_To_Transmitter_Buffer(Hold);


..../* Add the actual Data part to the checksum and transmitter */
   for (Data_Counter = 0; Data_Counter < DataCount; Data_Counter++)
      {
      CS += *Data;
      Add_Character_To_Transmitter_Buffer(*Data++);
      }



   /* Finally, create the checksum and transmit */
   CS = (0 - CS) - 1;
   Add_Character_To__Transmitter_Buffer(CS);


..../* Update buffer admin if necessary */
   SCIDataTransmitterBuffer__Buffer_Contents_Size += (Count + 6);
   }
}
```

Message decoder:

```
signed char CommCheckSumHold;
unsigned char CommCharacterCount;

/********************************************************/
unsigned char Communication_Receiver_Interpreter(unsigned char CharHold)
{
switch(CommInterpStage)
   {
   case SCICOMM__SEARCHING_FOR_START_CHARACTER:
      {
      if (CharHold == SCICOMMSTARTCHARACTER)
         {
         CommCheckSumHold = SCICOMMSTARTCHARACTER;
         CommInterpStage = SCICOMM__RECEIVING_DATA_COUNT;
         }
      break;
      }
   case SCICOMM__RECEIVING_DATA_COUNT:
      {
      CommCheckSumHold += (signed char)CharHold;

      CommCharacterCount = CharHold;

      /* Check data part size, take in account start char, char count, 3 byte ID and check sum */
      if (CommCharacterCount > (COMMCOMMANDDATABUFFERSIZE + 6))
         {
         /* Command data part is larger than buffer size, error */
         CommInterpStage = SCICOMM__SEARCHING_FOR_START_CHARACTER;
         }
      else
         {
         CommInterpStage = SCICOMM__RECEIVING_ID_NUMBER;
         CommCommandID = 0;
         CommCommandIDIndex = 1;
         }
      break;
      }
   case SCICOMM__RECEIVING_ID_NUMBER:
      {
      CommCheckSumHold += (signed char)CharHold;
      *((unsigned char *)&CommCommandID + CommCommandIDIndex) = CharHold;

      if (CommCommandIDIndex == 3)
         {
         CommCommandDataIndex = 0;
         CommCharacterCount -= 6;
         CommInterpStage = SCICOMM__RECEIVING_DATA;
         }
      else CommCommandIDIndex++;
      break;
      }
   case SCICOMM__RECEIVING_DATA:
      {
      CommCheckSumHold += (signed char)CharHold;
```

```
        CommCommandData[CommCommandDataIndex++] = CharHold;

        if (--CommCharacterCount == 0)
           {
           CommInterpStage = SCICOMM__VERIFYING_CHECKSUM;
           }
        break;
        }
     case SCICOMM__VERIFYING_CHECKSUM:
        {
        CommCheckSumHold += (signed char)CharHold;
        CommCheckSumHold++;

        if (CommCheckSumHold == 0)
           {
           /* Message is good, execute */
           Communication_Command_Execute();
           }

        CommInterpStage = SCICOMM__SEARCHING_FOR_START_CHARACTER;
        break;
        }
     }

  return CommInterpStage;
  }
```