Using ChamSys Remote Ethernet Protocol on MagicQ

Introduction

The MagicQ console and MagicQ PC software supports the use of an Ethernet protocol for controlling external devices, such as media servers, video or automation computers.

The protocol can also be used to remote control the MagicQ console or MagicQ PC software via a simple set of text commands.

Note that the use of the ChamSys Remote Ethernet Protocol on MagicQ PC is only enabled when it is connected to a MagicQ PC Wing.

On MagicQ commands are placed in the Macro field of the Cue Stack and are transmitted when the Cue starts to execute. In addition MagicQ will accept commands received according to a pre-defined protocol.

ChamSys Remote Ethernet Protocol

Enable ChamSys Remote Ethernet Protocol in the View Settings view of the Setup Window.

ChamSys Remote Ethernet Protocol is an Open Protocol – i.e. you do not need permission to use it for your own purposes. It is a UDP/IP based protocol using port 6553 in broadcast mode.

The structure of the UDP/IP packets are:

long32 chamsys;
word16 version;
byte seq_fwd;
byte seq_bkwd;
word16 length;
byte data;

where long32 is 4 bytes, word16 is 2 bytes and byte is 1 byte.

ChamSys is 4 characters C R E P. Note that on MagicQ this is stored as little-endian, so that on the network it will appear as P E R C.

The version is initially 0 and allows for future expansion of the protocol.

The fwd sequence number is an incrementing sequence number. It enables the receiving end to determine if packets are missed. In addition the receiving end should sends back the last sequence number it received in the backward sequence number.

Length is the length of the data field. It does not include the length of the ChamSys header.

Writing to the Ethernet port

Commands are transmitted from the Ethernet port by placing the command in the Macro field of the Cue Stack window (use Page Right to find the Macro field). In the View Settings view of the Setup Window, set the Ethernet Remote Protocol to “ChamSys Rem tx”.

The format of Ethernet commands is Y followed by the data. To send Ethernet data, the Y command must be the only macro command in the macro field. The Y command is followed by ASCII data contained within “ “ or ‘ ‘ or by decimal values separated by commas. For example to send Hello World followed by a carriage return:

Y”Hello World”,10,13

To send the hexadecimal data stream 00 01 02 03 04

Y0,1,2,3,4

To send text only:

Y”abcedf”

To send several lines of text:

Y”Hello”,10,13,”World”,10,13

On the Ethernet the data above is encapsulated in the data field of ChamSys Ethernet Remote Protocol in the UDP packet.

Note that commas ‘,’ are not allowed within the ASCII data inside “ “ or ‘ ‘ . If you wish to send ‘,’ then you must send it as its hexadecimal ASCII code.

Reading from the Ethernet

By default data received on the Ethernet is ignored. This can be changed to make MagicQ accept remote commands received on the Ethernet port. In the View Settings view of the Setup Window, set the Ethernet Remote Protocol to "ChamSys Rem rx".

ChamSys Remote protocol consists of simple commands consisting of a list parameter values separated by commas ',' and ending in a character A to Z (or a to z). Commands can contain spaces, tabs, and carriage returns; they are all ignored.

The commands are:

<playback number> A	Activate playback
<playback number> R	Release playback
<playback number> T	Test playback (activate with level 100%)
<playback number> U	Un-test playback (release with level 0%)
<playback number> G	Go on playback
<playback number> S	Stop (go back) on playback
<playback number> , <level> L	Set playback fader level
<playback number> , <cue id> , <cue id dec>J	Jump to Cue Id on playback
<page number> P	Change page
<channel number> , <level> I	Set intensity channel to level
<program command number> H	Remote programming command

The following parameter values are supported

<playback number> is a number between 1 and 10
<level> is an integer between 0 and 100.
<page number> is an integer between 0 and 100
<channel number> is an integer between 1 and 6144
<cue id> is an integer between 1 and 65536
<cue id dec> is an integer between 0 and 99

So for example, to set dimmer channel 4 to 50% you would use:

4,50I

To jump to Cue id 2.5 on playback 8 you would use:

8,2,5J

Commands can be sent back to back – e.g.

1A2A1S2G3,4I

Remote programming commands

Remote programming commands enable simple programming actions to be carried out from a remote terminal. Remote programming commands consist of the program command number followed by parameters and completed with an "H".

The commands are:

<01> , <start head> , [<end head>] H	Select one or more heads
<02> , <start head> , [<end head>] H	Deselect one or more heads
<03> H	Deselect all heads
<04> , <group number> H	Select group
<05> , <level> , [<time>] H	Set intensity of selected heads
<06> , <attribute number> , <value> , [<time>] H	Set attribute value of selected heads
<07> , <attribute number> , <value> , [<16bit>] H	Increase attribute of selected heads
<08> , <attribute number> , <value> , [<16bit>] H	Decrease attribute of selected heads
<09> H	Clear programmer
<10> , <palette id> H	Include position palette
<11> , <palette id> H	Include colour palette
<12> , <palette id> H	Include beam palette
<13> , <cue id> H	Include cue
<19> H	Update
<20>, <palette id> H	Record position palette
<21> , <palette id> H	Record colour palette
<22> , <palette id> H	Record beam palette
<23> , <cue id> H	Record cue
<30> H	Next head
<31> H	Previous head
<32> H	All heads
<40> H	Locate
<41> H	Lamp on
<42> H	Lamp off
<43> H	Reset

[] indicates an optional parameter
<level> is an integer between 0 and 100
<palette id> is an integer between 1 and 1024
<cue id> is an integer between 1 and 5000
<16 bit> is a flag. 0 for change in 8 bit resolution, 1 for change in 16 bit resolution
<time> is an integer time in seconds
<group number> is an integer between 1 and 200
<start head> and <end head> are integers between 1 and 6145

Attribute numbers

Intensity attributes (I1)

- -

-	-
-	-
-	Intensity (0)

Position attributes (P1)

Pos1 (46)	Pos5 (50)
Pos2 (47)	Pos6 (51)
Pos3 (48)	Pan (4)
Pos4 (49)	Tilt (5)

Colour attributes (C1)

Cyan (16)	Col4 (27)
Magenta (17)	Col3 (26)
Yellow(18)	Col2 (7)
Col mix (19)	Col1 (6)

Beam attributes page 1 (B1)

Shutter (2)	Rotate2 (11)
Iris (3)	Rotate1 (10)
Focus (12)	Gobo2 (9)
FX1 (14)	Gobo1 (8)

Beam attributes page 2 (B2)

Frost1 (32)	Rotate4 (31)
Frost2 (33)	Rotate3 (30)
Zoom (13)	Gobo4 (29)
FX2 (15)	Gobo3 (28)

Beam attributes page 3 (B3)

Macro1 (22)	FX8 (39)
Macro2 (23)	FX7 (38)
FX3 (34)	FX6 (37)
FX4 (35)	FX5 (36)

Beam attributes page 4 (B4)

Cont1 (20)	Cont8 (45)
Cont2 (21)	Cont7 (44)
Cont3 (40)	Cont6 (43)
Cont4 (41)	Cont5 (42)

Sample code fragments

The code fragments below show you could connect to MagicQ using simple C programming.

```
// ChamSys Ethernet remote protocol

#define REMOTE_ETHER_PORT  0x1999
#define MAX_CREP_MSG 1000

typedef struct {
  long32 chamsys;
  word16 version;
  byte seq_fwd;
  byte seq_bkwd;
  word16 length;
  byte data;
} remote_ether_message_t;


int remote_ether_sock = 0;
word16 remote_ether_fwd = 0;
word16 remote_ether_bkwd = 0;


int remote_ether_init(void)
{
  struct sockaddr_in name;

  char opts[100];
  socklen_t optlen = 100;
  int flags;
  int i;

  // For Windows OS we need to start winsocket

  #ifndef LINUX
  {
    WSAData ws;
    int code;
    code = WSAStartup(MAKEWORD(1,1),&ws);
  }
  #endif

  if (remote_ether_sock)
  {
    return (TRUE);
  }

  remote_ether_sock = socket (PF_INET, SOCK_DGRAM, 0);

  getsockopt (remote_ether_sock,SOL_SOCKET,SO_REUSEADDR, opts, &optlen);
  opts[0] = 1;
  setsockopt (remote_ether_sock,SOL_SOCKET,SO_REUSEADDR, opts, optlen);

  /* Give the socket a name. */

  name.sin_family = AF_INET;
  name.sin_port = htons (REMOTE_ETHER_PORT);
  name.sin_addr.s_addr = htonl (INADDR_ANY);
  if (bind (remote_ether_sock, (struct sockaddr *) &name, sizeof (name)) < 0)
```

```
  {
    closesocket(remote_ether_sock);
    return (FALSE);
  }

  getsockopt (remote_ether_sock,SOL_SOCKET,SO_BROADCAST, opts, &optlen);
  opts[0] = 1;
  setsockopt (remote_ether_sock,SOL_SOCKET,SO_BROADCAST, opts, optlen);

  {
  u_long block;
  block = 1;
  ioctlsocket(remote_ether_sock,FIONBIO,&block);
  }

  return TRUE;
}


int remote_ether_rx(char *data, word16 size)
{
  char message[MAX_CREP_MSG];
  int nbytes;
  remote_ether_message_t *rem = (remote_ether_message_t *) message;
  struct sockaddr_in name;
  int name_len = sizeof(name);

  if (!remote_ether_sock) return (0);

  nbytes = recvfrom (remote_ether_sock, message, MAX_CREP_MSG, 0, (struct
sockaddr *) &name, &name_len);

if (nbytes > 0)
  {
    if (rem->chamsys == (('C'<<24)|('R'<<16)|('E'<<8)|('P')))
    {
      int len = wswap(rem->length);
      remote_ether_bkwd = rem->seq_fwd;
      if (len<(MAX_CREP_MSG-(sizeof(remote_ether_message_t)+1)))
      {
        if (len>size) len = size;
        memcpy(data,&(rem->data),len);
        return (len);
      }
    }
  }

  return (0);
}

char remote_ether_tx(char *data, word16 size)
{
  // Format the message

  byte message[MAX_CREP_MSG];
  remote_ether_message_t *rem = (remote_ether_message_t *) message;
  int nbytes;

  struct sockaddr_in name;

  if (!remote_ether_sock) return (FALSE);
```

```
  if (size>(MAX_CREP_MSG-sizeof(remote_ether_message_t)+1))
  {
    size = MAX_CREP_MSG-sizeof(remote_ether_message_t)+1;
  }

  rem->chamsys = (('C'<<24)|('R'<<16)|('E'<<8)|('P'));
  rem->version = wswap(0);
  rem->seq_fwd = remote_ether_fwd;
  rem->seq_bkwd = remote_ether_bkwd;
  rem->length = wswap(size);
  memcpy(&(rem->data),data,size);

  my_broadcast_address.s_addr = ip_address | ~subnet_address;

  name.sin_family = AF_INET;
  name.sin_port = htons (REMOTE_ETHER_PORT);
  name.sin_addr.s_addr = dwswap (my_broadcast_address.s_addr);

  nbytes = sendto (remote_ether_sock, message, size +
(sizeof(remote_ether_message_t)-1), 0,
                (struct sockaddr *) & name, sizeof(name));

  if (nbytes>0) remote_ether_fwd++;

  return (TRUE);
}
```

Any comments on how these instructions could be improved will be gratefully received at support@chamsys.co.uk,