December 1, 2008

UMD and Tally Data Protocol Reference Document

V2

December 1, 2008

Contents

1 Revision History

Document Revision	Date	Notes
1	October 15, 2006	
2	December 1, 2008	Added multi-line support %S command

2 Introduction

This document describes the format of the protocol used to communicate text strings (UMD: under monitor display labels) and VGPI (Virtual General Purpose Input) data to Evertz Multi-Viewer devices for the purpose of video input labeling and tally. The remainder of the document will refer to both the MVP display cards and the VIP hardware as devices.

3 Protocol Format for UMD data

UMD data is conveyed using the following command format (entirely in ASCII):

Basic Commands:

%14D%1STEST%Z

where:
%14D specifies the UMD ID or protocol ID (a unique protocol ID is assigned to each UMD)
%1S specifies the line (the default is 1), see below for advance use case
TEST specifies the text that is to be displayed in the UMD (max. 16 characters)
%Z specifies the end of the message

Advanced Commands:

Colour:
%22D%1S%85CSOURCE1%Z

where:
%85C specifies the colour red

%38D%1S%170CCAMERA%Z
where:
%170C specifies the colour green

%244D%1S%255CWORLD%Z
where:
%255C specifies the colour amber

Justification:
%7D%1SLEFT%1JCENTER%2JRIGHT%Z
where:
%1J specifies center justification (text will be left justified if no justification stated)
%2J specifies right justification

Line:
%1D%1Stest%Z
Where:
%1S specifies the first line, first UMD under the same PID

%1D%2Stest%Z
Where:
%2S specifies the second line, second UMD under the same PID

%1D%4Stest%Z
Where:
%4S specifies the third line, third UMD under the same PID

%1D%7Stest%Z
Where:
%7S specifies all three lines, all three UMDs under the same PID will be addressed

4 Protocol Format for VGPI data

Virtual GPI data is conveyed using the following command format (entirely in ASCII):

%16S<virt_gpi_num>=<0 | 1>%Z

where:

%16S specifies the use of the virtual GPI command
<virt_gpi_num> is the virtual GPI number, 1 – 320
<0> indicates the virtual GPI is off
<1> indicates the virtual GPI is on
%Z is an end delimiter for the message

As examples:

%16S1=0%Z (virtual GPI 1 is off)
%16S1=1%Z (virtual GPI 1 is on)
%16S42=0%Z (virtual GPI 42 is off)
%16S42=1%Z (virtual GPI 42 is on)

5 Enabling the Devices to Receive Data

There are 2 ways a device can receive data:
Over serial
Over TCP/IP

Each of these is configured using the test/debug com port of the display card.

5.1 Using Serial

From the display card's Main Menu select Under Monitor Display Setup.
Select Protocol Select.
Select Image Video as the UMD protocol.
Select Serial as the input type.
Perform Save and Exit.
Select auxiliary serial port setup
Configure baud rate, data bits, stop bits, and parity to match the transmit devices serial settings
Perform Save and Exit
Reboot the display card for changes to be applied

5.2 Using TCP/IP

From the display card's Main Menu select Under Monitor Display Setup.
Select Protocol Select.
Select Image Video as the UMD protocol.
Select Network as the input type.
Enter the TCP port over which the display card will receive data . Typically 9801 is used.
Perform a Save and Exit.
Reboot the PPV.

Note: the devices use a single serial port or TCP/IP port to communicate to upstream devices. Therefore, they can only accept a single connection at a time, in the case of the TCP/IP interface if a second device tries to attach to this port it will bump the previous connection. This is done so that the port will not be left in a locked state, and that any retries to connect to the port will always result in a successful connection.

6 Network Application Examples

6.1 Windows

6.1.1 Sample Code

The following C code provides an example of a Windows console application used to send VGPI data to a device. This code makes use of wsock32.lib.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <winbase.h>
#include <winsock.h>

void showUsage(void)
{
   printf("usage: vgpi dst_ip dst_port virt_gpi val\n");
   printf("\n");
   printf("where: dst_ip   = IP of peer to receive virtual GPI data\n");
   printf("       dst_port = TCP port over which data will be sent\n");
   printf("       virt_gpi = virtual GPI number (1 - 320)\n");
   printf("       val      = virtual GPI value (0, 1)\n");
}

int main(int argc, char **argv)
{
   SOCKADDR_IN  peer;
   LPHOSTENT    lpHostEntry;
   SOCKET       s;
   WORD         wVersionRequested;
   WSADATA      wsaData;
   char         *pDstIp;
   char         txBuf[25];
   int          virtGpi;
   int          dstPort;
   int          val;
   int          nRet;
   int          retVal;

   /* verify parameters */
   if (argc < 5)
   {
      showUsage();
      goto exit;
   }

   pDstIp = argv[1];
   dstPort = atoi(argv[2]);
   virtGpi = atoi(argv[3]);
   val = atoi(argv[4]);

   if (dstPort == 0)
   {
      showUsage();
```

```
    goto exit;
}

if ((virtGpi < 1) || (virtGpi > 320))
{
    showUsage();
    goto exit;
}

if ((val != 0) && (val != 1))
{
    showUsage();
    goto exit;
}

/* we'd like to use sockets version 1.1 */
wVersionRequested = MAKEWORD(1,1);
if (WSAStartup(wVersionRequested, &wsaData) != 0)
{
    printf("error: unable to start winsock\n\n");
    goto exit;
}

if (wsaData.wVersion != wVersionRequested)
{
    printf("error: unable to start winsock version 0x%04x\n\n", wVersionRequested);
    goto cleanSocket;
}

/* create a TCP/IP stream socket */
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET)
{
    printf("error: unable to create socket\n");
    goto cleanSocket;
}

/* fill in the details of our peer */
lpHostEntry = gethostbyname(pDstIp);
if (lpHostEntry == NULL)
{
    printf("error: unable to locate %s\n", pDstIp);
    goto cleanSocket;
}

peer.sin_family = AF_INET;
peer.sin_addr   = *((LPIN_ADDR)*lpHostEntry->h_addr_list);
peer.sin_port   = htons((unsigned short)dstPort);

/* connect to our peer */
if ((connect(s, (LPSOCKADDR)&peer, sizeof(struct sockaddr))) == SOCKET_ERROR)
{
    printf("error: error %d occurred connecting to socket\n\n", WSAGetLastError());
    goto closeSocket;
}
```

```
   /* prepare and send virtual GPI data to our peer */
   sprintf(txBuf,"%%16S%d=%d%%Z", virtGpi, val);
   printf("%s\n", txBuf);
   nRet = send(s, txBuf, strlen(txBuf), 0);
   if (nRet == SOCKET_ERROR)
   {
      printf("error: error %d occurred sending to socket\n\n", WSAGetLastError());
      goto closeSocket;
   }

   printf("success\n");
   retVal = 0;
   sleep(1000);

closeSocket:
   closesocket(s);

cleanSocket:
   WSACleanup();

exit:
   return (0);
}
```

6.1.2 Usage

Suppose a device with IP address 192.168.18.32 is listening on port 9801. To use the application code listed above to turn on VGPI 21 one would execute (assuming vgpi.exe is the name of the executable):

```
vgpi   192.168.18.32   9801   21   1
```