VideoCom Developer Information

1 Introduction

This information is based on the following firmware versions and will be changed in the future.

VideoCom	Cat. number	Firmware version
VideoCom	337 47	1.20 or higher
VideoCom USB	337 47 USB	2.00 or higher

You have several possibilities to develop software for VideoCom. The simplest way is to use our Delphi/Lazarus component VideoComSerial for Windows and Linux (see section 2) which includes the access over the USB port and the serial port.

If you prefer another programming language, you may use our VideoComAPI.DLL (Windows) or libvideocomapi.so (Linux) to access VideoCom (see section 3). The most basic way is to use the USB and Serial protocol of VideoCom (see section 4).

2 Delphi/Lazarus Component VideoComSerial

Before using the VideoComSerial component under Windows, you have to install the VideoComSerial component in Delphi. In Delphi 2, the components usbserial.pas and cassyserial.pas are installed by “Component -> Install -> Add”. In Delphi 3 or higher, open and install the package ldusbserial.dpk using "File -> Open".

For Linux installation, you must use Lazarus with the Free Pascal Compiler Version 2.0 or higher. Open and install the package ldusbserial.lpk using "Components -> Open package file". For installing a package in Lazarus, you have to recompile the IDE.

In Delphi as in Lazarus you will get a tab called “LD Didactic”, which shows the VideoComSerial component.

Important Linux notes: VideoCom USB is supported since kernel 2.6.13. However, make sure, that you have `rw` access to `/dev/ttyS0` (serial port) or `/dev/ldusb0` (USB port). This can be done by copying `48-ldusb.rules` to `/etc/udev/rules.d` (be sure to be member of the group `users`). For Lazarus' thread support (used for VideoCom USB), you have to include `CThreads` as first unit in your applications' uses clause.

The VideoComSerial component has the published properties `CommPort` and `USB` and the event `OnDevNodesChanged` (Windows only), which you may change in the object inspector of Delphi or Lazarus. The other properties or functions are useable during runtime only.

Some properties are read-only. They are marked with a small **R**. Please be aware of it.

2.1 Finding VideoCom

```
   Property CommPort : String;
   Property USB : Boolean;
R  Property Open : Boolean;
```

are used to define the port of VideoCom (e.g. `CommPort:='COM1'` for the serial port 1 or `USB:=True` for the USB port) and to know its status.

```
R  Property Version : String;
```

The property `Version` reads a string `VideoCom x.xx` with the actual firmware version of the attached VideoCom in the format `x.xx`.

2.2 Reading Data

```
TValueArray   = Array [0..2047] Of Integer;
TVideoComData = Record
                  Running     : Boolean;
                  PeriodTime  : Integer;  // in 1.25 ms
                  ExpTime     : Integer;  // exp. time for intensities
                  LEDAuto     : Boolean;  // LED time automatic
                  LEDTime     : Integer;  // in 0.125 ms
                  RecTime     : Integer;  // in 1.25 ms
                  Count       : Integer;  // number of values in array
                  Values      : TValueArray;
                  DataWaiting : Boolean;  // next data already waiting
                End;

Function ReadData(Var VideoComData : TVideoComData) : Boolean;
```

`ReadData` reads the actual data from VideoCom and returns immediately. It returns `True` if data was ready and `False`, if data was not ready. In this case, just try it a little bit later. See the following table for the return values of the parameters.

`Running`	status of the START/STOP push button of VideoCom
`PeriodTime`	time difference between two position measurements in units of 1.25 ms (or 0 for intensity measurements)
`ExpTime`	exposure time of one intensity measurement if it is initiated with the `J` command (see section 4)
`LEDAuto`	`True`, if LED time is controlled automatically
`LEDTime`	on-time for the LED flash in units of 125 µs (max. 10*125 µs = 1.25 ms)
`RecTime`	position recording time after starting the measurement
`Count`	number of valid values in the `Values` array
`Values`	measurement values
`DataWaiting`	`True`, if new positions are already waiting. Intensities are skipped if not read fast enough.

2.2 Writing Data

```
Procedure WriteString(Command : String);
```

`WriteString` controls VideoCom by software. For the available commands refer to section 4.2.

Examples

```
WriteString('S10A');   // starts position measurement with dt = 10*1.25 ms

WriteString('J15');    // starts intensity measurement with 2048 pixels
```

3 Using VideoComAPI.DLL or libvideocomapi.so

If you prefer another programming language, you may use our library VideoComAPI.DLL (Windows) or libvideocomapi.so (Linux). The videocomapi.h C-header file defines all included definitions and functions, which are very similar to the already described Delphi/Lazarus component VideoComSerial. For your reference, you also find some pascal source code in VideoComAPI.dpr.

Before using any function of VideoComAPI.DLL or libvideocomapi.so, it needs to be initialized with

```
void VideoCom_Init(char* COM);
```

with `COM='USB'`, `COM='COM1'`, etc.. Before closing your application, the library wants to get a

```
void VideoCom_Exit();
```

call. Every function sets an internal error flag, which you can access with

```
TVideoComError VideoCom_GetLastError();
```

Most of the described properties or function of section 2 can be accessed with an appropriate `VideoCom_...` call. If a function call fails, `VideoCom_GetLastError()` returns an error.

Example for reading positions (see also the example positions.c)

```
VideoCom_Init("USB");       // initializes DLL and opens VideoCom USB port
if (VideoCom_GetOpen()) {
  bool Running,LEDAuto,DataWaiting;
  int PeriodTime,LEDTime,RecTime,Count;
  TValueArray Positions;
  VideoCom_WriteString("S10A");  // start with time interval of 12.5 ms
  for (int i=0; i < 100; i++) {
    while (!VideoCom_ReadPositions(&Running,&PeriodTime,&LEDAuto,&LEDTime,
                                   &RecTime,&Count,Positions,&DataWaiting)
           && VideoCom_GetLastError() == veOK);
    /*
    save Positions at Time RecTime*1.25 ms
    */
  }
  VideoCom_Exit();
}
```

Example for reading intensities (see also the example intensities.c)

```
VideoCom_Init("USB");       // initializes DLL and opens VideoCom USB port
if (VideoCom_GetOpen()) {
  bool Running,DataWaiting;
  int PeriodTime,ExpTime,LEDTime,RecTime,Count;
  TValueArray Intensities;
  VideoCom_WriteString("J15");  // measure 2048 intensities
  while (!VideoCom_ReadIntensities(&Running,&PeriodTime,&ExpTime,&LEDTime,
                                   &Count,Intensities,&DataWaiting) &&
         VideoCom_GetLastError() == veOK);
  /*
  display measured intensities
  */
}
VideoCom_Exit();
```

4 USB and Serial Protocol

VideoCom and VideoCom USB are using a similar protocol to receive a command and to transmit the answer.

If you use the USB port, please open a HID device (Windows) or /dev/ldusb0 (Linux since kernel 2.6.13) with Vendor ID $F11 and Product ID $1200. For sending and receiving data, you must use Output Reports and Input Reports with the Report ID 0 (Windows only). Every Output/Input Report consists of the report ID (=0, Windows only) and 8/32 additional data bytes. However, not all of the 8/32 data bytes must be valid data. If the first data byte equals 7/31 or less, this byte represents the length of the following valid data. If the first data byte equals 8/32 or higher, all 8/32 data bytes are valid data (including the first data byte itself).

USB Output Report example (Windows):

$00 $04 'S' '1' $13 $10 $00 $00 $00 $00 will be interpreted as 'S' '1' $13 $10

USB Output Report example (Linux):

$04 'S' '1' $13 $10 $00 $00 $00 $00 will be interpreted as 'S' '1' $13 $10

VideoCom USB must be initialized with a special Output Report before data can be sent or received:

USB Initialization Output Report (Windows):

$00 $00 $00 $FF $00 $00 $00 $00 $00 Report ID 0, Length 0, Status request 0, Init $FF

USB Initialization Output Report (Linux):

$00 $00 $FF $00 $00 $00 $00 $00 Length 0, Status request 0, Init $FF

If you use the serial port RS232, please configure it to 19200 baud, 8 data bits, no parity, no handshake. Every byte sent will be interpreted as valid data.

Every data line VideoCom sends to the computer ends with CR/LF (13/10). Every command you send to VideoCom has to end with CR/LF, too. Instead of sending commands to VideoCom, you may also use the VideoCom push buttons.

4.1 Receiving Data from VideoCom

VideoCom has two operating modes: position and intensity.

In **position mode**, VideoCom sends ASCII lines (no binary data) beginning with `S`:

`Sabbcdeeeexxxyyy...` +CR/LF

`a` `0`: STOP (time stopped) , `1`: START (time running)
measured positions are also transmitted if VideoCom is stopped
(you may use it for monitoring the motion without storing the positions)

`bb` Δt in 1.25 ms between two measurements
`bb` is as two-digit hexadecimal number (e. g. `0A`: Δt = 10*1.25 ms = 12.5 ms)

`c` `0`: automatic exposure off (t_{LED} constant), `1`: automatic exposure on (t_{LED} calculated)

`d` LED on time t_{LED} in 10 % of 1.25 ms (= full exposure time)
`d` is a hexadecimal digit (e. g. `A`: t_{LED} = 10*10 % = 100 % = 1.25 ms)

`eeee` time t in 1.25 ms since measurement started or `0000` if measurement stopped
`eeee` is a four-digit hexadecimal number (e. g. `0320`: t = 800*1.25 ms = 1000 ms)

`xxx` position x_1 (0 to 4195) of the first recognized body or end of the line (no body recognized)
`xxx` is a three-digit hexadecimal number (e. g. `800`: x_1 = 2048 - the middle of the CCD)

`yyy` position x_2 of the second recognized body or end of the line

`...`

One single `S`-line may contain up to 20 positions x_i of recognized bodies. Since the serial transmission time of long lines may exceed the chosen Δt, measurement lines may be skipped (e. g. 4 bodies will result in a 24 byte line including CR/LF, which will take more than Δt = 12.5 ms for transmission and every second `S`-line will be skipped). USB transmission is much faster and will not show this problem.

In **intensity mode**, VideoCom sends mixed ASCII and binary lines beginning with `I` or `J`:

`Iabdeexx...xx` +CR/LF or
`Jabcdeexx...xx` +CR/LF (new with firmware 1.2)

`a` `0`: STOP, `1`: START
measured intensities are also transmitted if VideoCom is stopped
(you may use it for monitoring the measurement without storing the intensities)

`b` $64*2^b$ intensities will be transmitted in single slices - every slice consists of
64 intensity bytes for VideoCom (serial) which can use `b`: `0..5` (64 up to 2048 intensities) or
512 intensity bytes for VideoCom (USB) which can use `b`: `3..5` (512 up to 2048 intensities)

`c` exposure time in 1.25 ms (up to firmware 1.1 the exposure time is 1.25 ms constant)
`c` is a hexadecimal digit (e. g. `A`: exposure time = 10*1.25 ms = 12.5 ms)

`d` LED on time t_{LED} in 10 % of 1.25 ms (= minimum exposure time)
`d` is a hexadecimal digit (e. g. `A`: t_{LED} = 10*10 % = 100 % = 1.25 ms)

`ee` number of intensity slice (between 0 and 2^b-1)
`ee` is a two-digit hexadecimal number (e. g. `14`: 20th intensity slice is appended)

`xx..xx` intensity slice (64 bytes binary data for serial, 512 bytes binary data for USB connection)

One single intensity slice is measured at the same time. Different slices are measured at different times. Therefore a complete measurement of all intensity slices needs a constant intensity distribution over a larger amount of time (up to a few seconds for up to 2^5 = 32 slices with serial connection).

4.2 Sending Data to VideoCom

VideoCom can be controlled by short ASCII commands transmitted to VideoCom. Those commands have the already described syntax.

`Sabbcd` +CR/LF

switches to position mode and changes the given settings and

`Iabd` +CR/LF or
`Jabcd` +CR/LF (new with firmware 1.2)

switches to intensity mode and changes the given settings. All commands can be truncated (e. g. `S1` starts the position measurement, Δt, automatic exposure and t_{LED} are not changed or `I05` stops the intensity measurement and switches to 2048 intensity values, exposure time and t_{LED} are not changed).

In addition, a simple `V` + CR/LF returns the VideoCom version in a single line like `VideoCom x.xx`. This version may be changed by updating the VideoCom firmware by the standard VideoCom software.

5 Further Information

For further information contact mhund@ld-didactic.de. We wish you a lot of fun developing software for VideoCom.

Available VideoCom software:

VideoCom (Windows)	http://www.ld-didactic.com/software/videocom.exe
LabVIEW driver (Windows+Linux)	http://www.ld-didactic.com/software/videocomlabview.zip

Dr. Michael Hund