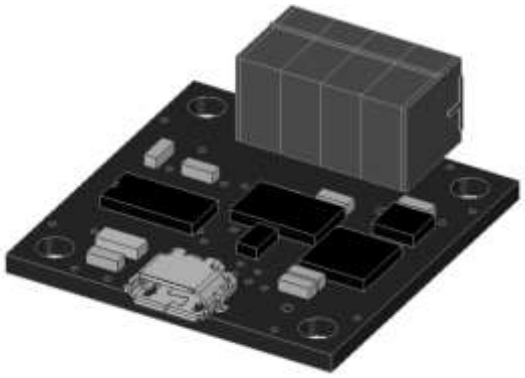
Linux USB-DV0 expansion boards controller

Features

- Allows Linux systems to manage input/output boards compatible with DV0 bus.
- Free available Linux commands and API library (Python, C++ or Java) for setting parameters and accessing expansion boards
- USB Powered, no need of external power supply
- Native Linux USB-serial emulation. No drivers required.
- Quick installation: only one program is required (dv0_manager), free downloading for several Linux available.



Technical specifications

- USB connector: mini-B
- DVO connector specs:
  - Socket reference: 20020004-D041B01LF from FCI.
  - Solid/Stranded wire: AWG 16 to AWG 26
  - Wire cross section: 1.5mm to 0.2mm
  - 5V output maximum current: bypassed from the Linux system
  - Data signal A and B ESD, EFT and Surge protection: IEC 61000-4-2 (ESD), IEC 61000-4-4 (EFT) and IEC 61000-4-5 (Surge)
- Up to 16 expansion boards management capability
- Command latency from user application to IO expansion boards: 15ms
- USB to Serial emulation chip: CP2102

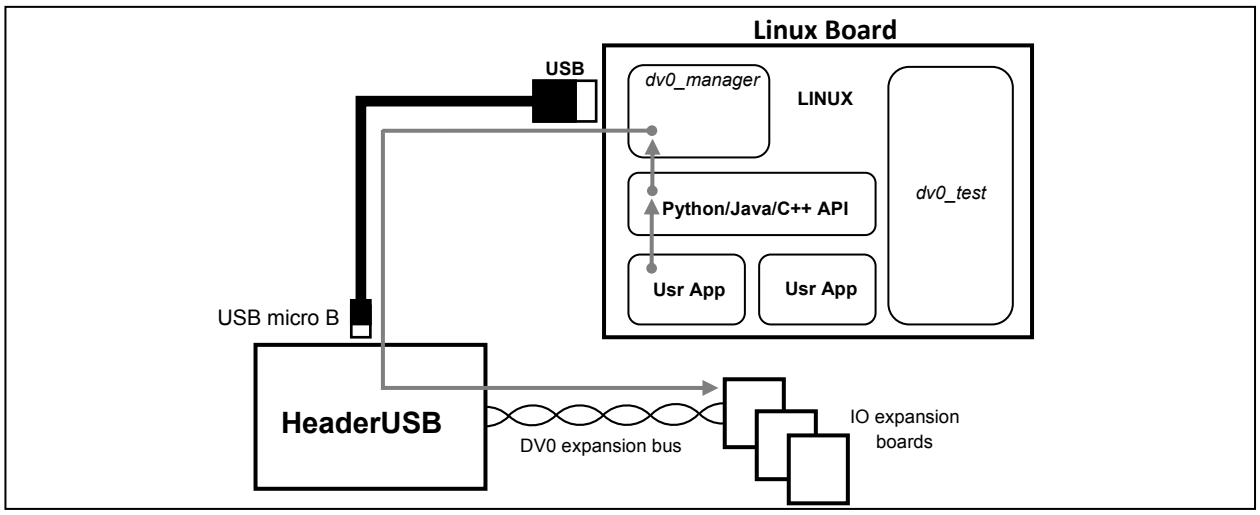


Fig. 1 Simplified block diagram

Table of contents

1. Functional overwiew 4
1.2.1 Discovery process 4
2. Electrical characteristics 5
2.1 Identifying connectors 5
2.1.1 DV0 Bus connector 5
2.2 Absolute maximum ratings 6
2.3 DVO Bus length 6
2.3.1 Data transmission degrading 6
2.3.2 Voltage drop 6
3. Software 8
3.1 Architecture description 8
3.2 *dv0_manager* installation 9
3.2.1 *dv0_manager* return values 9
3.3 *dv0_manager* command line parameters 10
3.3.1 Parameter *-serial_port <port>* 10
3.3.2 Parameter *-port <port number>* 10
3.3.3 Parameter *-ip_address <ip_address>* 10
3.3.4 Parameter *-login <login name>* 10
3.4 *dv0_test* command line parameters 10
3.4.1 Parameter *-ip_address <ip address>* 11
3.4.2 Parameter *-port <Port number>* 11
3.4.3 Parameter *-login <login name>* 11
3.4.4 Parameter *-password <password>* 11
3.4.5 Parameter *-get_manifest <address>* 11
3.4.6 Return values of *dv0_test* 11
3.5 *DV0* API 12
3.5.1 Creating an instance of DV0 class 12
3.5.2 Connecting to *dv0_manager* 13
3.5.3 API functions related to HeaderUSB 14
3.5.4 API examples 15
4. Mechanical drawings 18
5. Accessories 18

Important notice.....19

Revision history

V1.0 Jan-2015

1. FUNCTIONAL OVERWIEW

The DeValirian HeaderUSB allows Linux systems to easily control and manage a chain of Input/Output expansion boards, giving easy control of physical inputs and outputs, like analog or digital sensors, relays, stepper motors, servo motors, keyboard, displays, etc

These boards need a 5V power supply and a RS422 twisted pair serial channel connected in parallel bus mode. Thanks to the address selector micro-switch located at each IO board, the header can send data and poll information from every board connected to the bus.

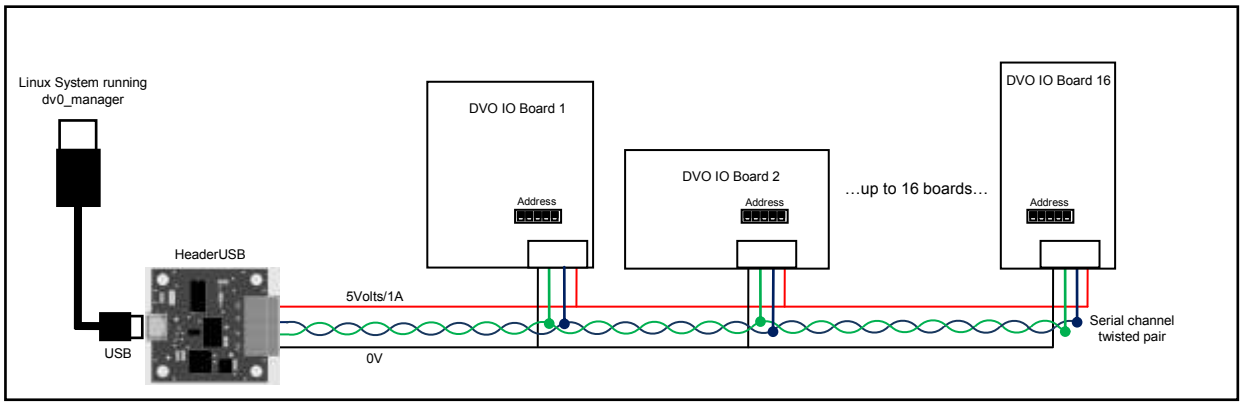


Fig. 2 DVO IO expansion board bus

After the First Wake Up state, the HeaderUSB control unit executes the initial discovering process by individually polling to all possible address. It takes roughly one second and, once it is finished, the HeaderUSB board knows how many boards are connected and what their address are.

User programs can read/write data from/to boards thanks to *dv0_manager* program which acts as a tunnel between user programs and IO boards. A complete and easy to use library is free available for C/C++, Java and Python. See Section 3 for more information and examples.

So, user programs connect to *dvo_manager*, which talks with the HeaderUSB control unit, the actual responsible for send and receive data through the serial channel. During idle states, that is, when user programs don't invoke any library function, the HeaderUSB control unit continuously polls to all known board to get the value of their inputs and maintains a local database with these values. Thanks to this database, user programs functions that want to read board's input values exhibit no extra delay independently of how many boards are connected.

Also, user program functions that want to write to a specific board are considered as a high priority procedure by the HeaderUSB control unit, and it only wait the current poll operation to launch the command to the specified board. This strategy ensures a maximum latency of 15 ms both for read and write operations from user programs.

1.2.1 Discovery process

The broad discovery process, that is, a sequential enquiry to all possible 63 address, is made whenever:

- The First Wake up occurs
- Every time that the HeaderUSB goes to On State from Idle State

An individual discovery process, that is, an enquire to a unique and specific address, is made whenever a user program wants to access to a board that is not present in the HeaderUSB control unit database. In order to maintain the latency time declared above, the control unit returns an error as "Board not present" to the user program function, but simultaneously sends a poll to this board. If this board is really connected, the next user program function to this board will be successful.

This allows a "hot connection" feature for the DV0 Bus but the user program procedure has to check every function return value and in case of "Board not present" error, retry the function call instead of treating this error as an exception.

2. ELECTRICAL CHARACTERISTICS

This section explains the electrical characteristics of each HeaderUSB connector and the length issues related to the DV0 bus cabling.

2.1 Identifying connectors

The Figure 3 shows the location of all connectors of HeaderUSB board

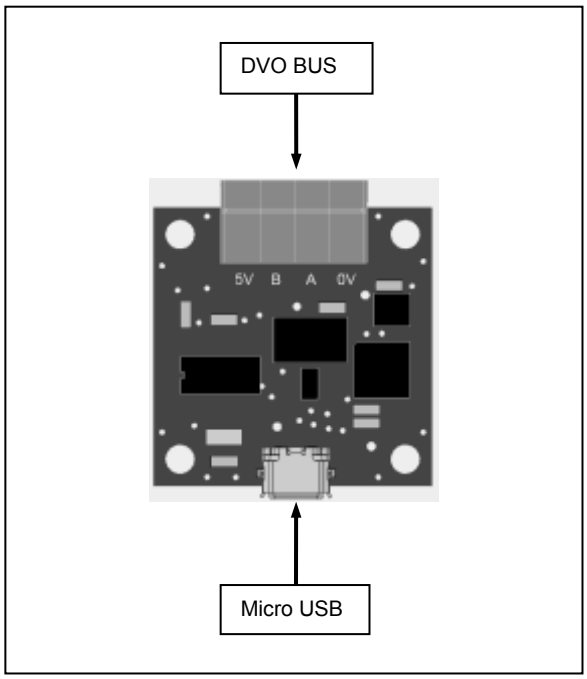


Fig. 3 HeaderUSB connectors

2.1.1 DV0 Bus connector

This receptacle connector contains power and data for IO expansion board. An extracting socket with screws is supplied with the HeaderUSB board to facilitate the installation of IO expansion board bus cabling (AWG 16..24, cross section 1.5 to 0.2 mm)

Technical data:

- Socket reference: 20020004-D041B01LF from FCI.
- Solid/Stranded wire: AWG 16 to AWG 26
- Wire cross section: 1.5mm to 0.2mm
- Data signal A and B ESD, EFT and Surge protection: IEC 61000-4-2 (ESD), IEC 61000-4-4 (EFT) and IEC 61000-4-5 (Surge)
- 5V output maximum current: 1A
- 5V output short-circuit current: 2A

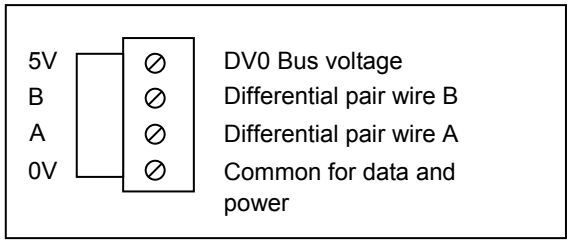


Fig. 4 DVO Bus connector pin out

Connect differential pair A to all A inputs of connected IO boards, and pair B to all B inputs as well. Also, connect the common data pin (0V) to all IO boards. The output marked as 5V is the DV0 bus voltage output. It is raised to 5V during the Power On Process and disconnected during the Power Off Process (see 1.1.5 and 1.1.3). This output can be used to fed the IO boards connected to the bus, but it is not mandatory since IO boards can be locally supplied, as explained at 2.4.2 Voltage drop.

All IO Boards PCB are marked with the legend shown at Figure 4.

2.2 Absolute maximum ratings

Absolute maximum ratings for the HeaderUSB board are listed below. Exposure to these maximum rating conditions for extended periods may affect device reliability. Functional operation of the device at these, or any other conditions above the parameters indicated in the operation listings of this specification, is not assured.

Operating Ambient temperature……....0°C to +70°C
Storage temperature .....-55°C to +125°C
Voltage on USB connector.....-0.1V to +5.5V
Maximum Current out to DV0 Bus……..…..1.2A

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

2.3 DVO Bus length

There are two different considerations about DV0 Bus wire length. On one hand, there is the data transmission degrading, on the other hand there is the issue of the voltage drop due to the resistance of the cable.

2.3.1 Data transmission degrading

Data is transmitted using a RS422 standard level of differential voltage at 100kbps on the wires marked as A and B at the DV0 Bus connector. This standard doesn't define a wire type to be used, in opposition to other standards like Ethernet or similar. For this reason there is not an official maximum cable length for this standard.

However, there are a conservative data based on empirical test that assures a 1000 meters length at 100 kbsp using a twisted pair of 15pf, 5ns/m of propagation speed.

To achieve the maximum noise rejection that RS422 offers, it is mandatory to use a twisted pair. We recommend a twisted pair of AWG 20 (0.5 mm^2) because test realized with this cable had shown no degradation over 50 meters bus with 16 IO boards connected (no termination resistors are required for that distance)

Even with the CRC that protects all frames which travel through the bus and despite the excellent performance shown over 50 meters bus length, we don't recommend to exceed this limit due to the ground common mode noise that could appear if the power wires of bus are not strong enough or there is a ground loop.

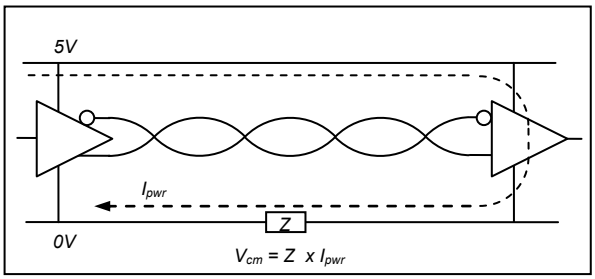


Fig. 5 Common mode noise

The common mode noise is generated by the power current when it flows throughout the power wires. As the power wires don't have a zero impedance, a common voltage appears (see Figure 5) and, if this noise is greater than 7 volts, the receiver can't decode data so the current frame will be ignored. Also, all strong electric or magnetic field close to the power wires can induce a common mode voltage as well.

To reduce the common mode voltage noise, it is strongly recommended:

- Twist the power wires as the data wires. This reduces the inductive factor of impedance and the magnetic fields disturbance.
- Avoid cabling power wires in parallel with other power lines, specially if they carry strong inductive loads, as motors or electro valves.

2.3.2 Voltage drop

The voltage drop across the power wires is due to the DC impedance (i.e., the resistance) of these wires. It is easy to calculate this voltage drop average value knowing the section of the cable, its length and the average current consumption of the IO board.

Knowing the section of the cable, manufacturers report its resistivity (ρ) in ohms/meter. Then, to obtain the voltage drop

$$V_{drop} = I_{avg} \times length \times \rho$$

Using the example 2 explained above, with AWG 20 section and 25 meters length between the HeaderUSB board and the IO board, the voltage drop will be

$$V_{drop} = I_{avg} \times length \times \rho = 4 \times 0.134 \times 25 \times 0.033 = 0.442\ [V]$$

In the above expression, the figure 0.033 is the nominal resistivity of AWG 20 wire (from manufacturer). As all DVO IO boards can withstand

this voltage drop over 5V, this example is correct. However, this voltage drop is near the limits so, if we plan a larger bus length, a new approach is needed.

To solve this issue, there are two solutions available:

- Increase the wire section
- Feed IO board with local power supplies

Increasing the wire section is the simplest solution but sometimes is not practical due to the cost increment and the difficulty of cabling thick wires on the small connectors of IO boards.

The use of local power supplies offers more flexibility and solves the problem of connecting heavy loads (as motors or servos) to the USB Bus power, witch is usually limited to 500mA.

Figure 6 shows this 3-wires approach. When using local powering, no high current flows through the ground line but there is the risk of creating a ground loop due to the difference of potential between the earth reference of the local power supplies. However, the common mode voltage generated in this case is far lower than the one due to power supply current.

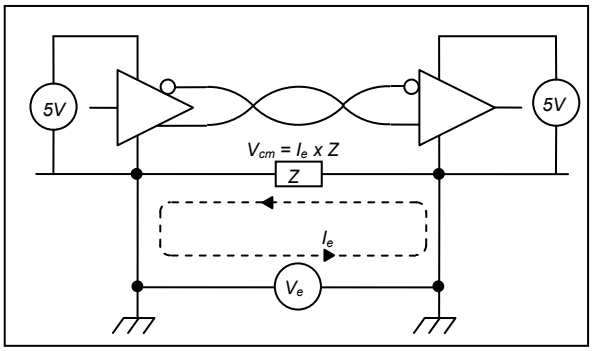


Fig. 6 Local powering and ground loop

CAUTION:

HeaderUSB and DV0 IO boards are protected against high transient voltage according to IEC 61000-4-5 but it is not recommended to deploy a large length of cable with different earths references: lighting can generate a sudden high energy transient that would destroy the transceivers despite the surge protections.

This approach is intended just to avoid large currents flowing through the power lines, not to increase the distance between HeaderUSB and IO board. If you want to, do it at your risk, but at least add a more powerful TVS protection and assure a good earth connection at both power supplies. Also, it is strongly recommended to avoid placing data lines close to power lines.

3. SOFTWARE

The HeaderUSB board comes with several free programs, API and examples of C, Python, Java and Linux scripts. Their goal is to configure the power up/down behavior and scheduling and to allow user programs to gain access of the real word reading and writing DV0 input/output boards.

These are all the free software parts related to HeaderUSB:

- The ***dv0_manager*** program: the one and only one who talks with the HeaderUSB control unit.
- The ***dv0_test*** utility: command program to test the *dv0_manager* and to send commands and settings to the HeaderUSB control board. Intended to be used in Linux scripts
- The ***DV0 API libraries*** in C, Java and Python. A full set of functions to control the HeaderUSB and the DV0 input/output board connected to the DV0 bus
- **Source code** examples written in C, Python and Java (Android)

All these parts and their associate documentation are free available in the Technical Information section of www.devalirian.com

3.1 Architecture description

Local architecture is the most common scenario and it is shown in the Figure 7.

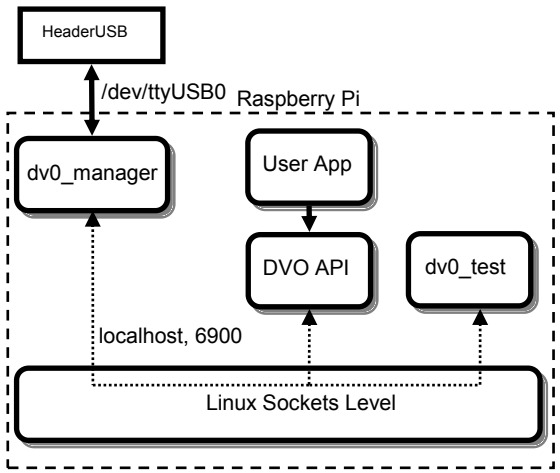


Fig. 7 Local architecture

In this case, the *dv0_manager* program waits for a socket connection listening to the interface "*localhost*" at port "6900". Both *dv0_test* program and user applications connect to *dv0_manager* using the function "*Open*", with that interface and port (which are the default values). The *dv0_manager* "tunnels" commands from *dv0_test* and user applications to the HeaderUSB control unit through the USB-serial emulation device *ttyUSB0 (*sometimes could be ttyUSB1 when connect and disconnect the USB cable). Note that all elements run inside the Linux system

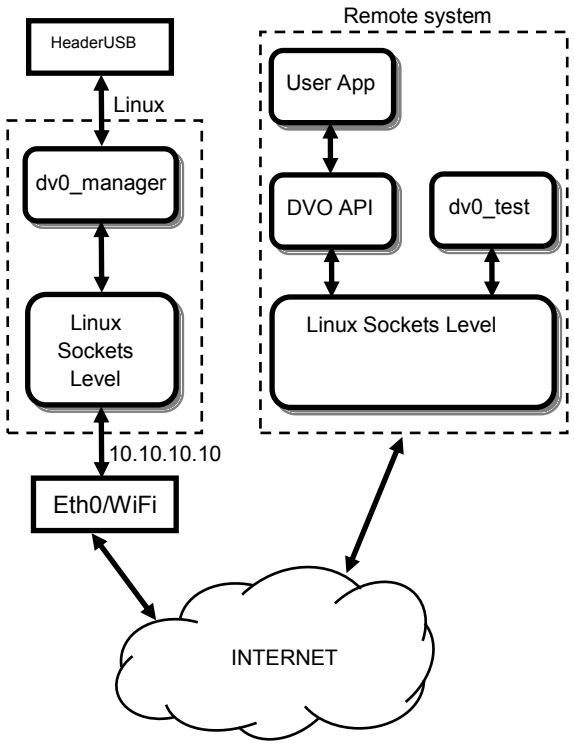


Fig. 8 Remote access architecture

Since the socket level physical layer is transparent to the user applications, it is possible to replicate the local architecture to a remote architecture, where the command and data traffic between *dv0_manager* and *DV0 API* or *dv0_test* travel through internet. Figure 8 shows this case.

In this example, the Linux system is connected to internet via an interface which IP address is 10.10.10.10, no matter if it is an Ethernet or a WiFi connection. To accept data and commands from this interface as their come from localhost, only a few initialization changes must be performed. More exactly, the *dv0_manager* must be called with the –*ip_address* command line parameter :

```
dv0_manager -ip_address 10.10.10.10
```

and the "Open" function of DVO API requires a string with that IP address:

```
Open("10.10.10.10")
```

Note that when using a remote connection, some security issues must be taken into account to avoid unexpected and unwanted connection. See 3.3 for more information on login restrictions for a remote access.

3.2 *dv0_manager* installation

Log into your Linux system as root, download *dv0_manager* and *dv0_test* from the Technical Information section of www.devalirian.com and place into some directory. For instance, create and use /home/dv0

Allow them execution permissions with *chmod* command:

```
chmod +x *
```

Edit the file */etc/rc.local* with your preferred Linux editor. For example:

```
nano /etc/rc.local
```

In almost all Linux distributions, this file is intended to contents the last commands that must be launched when the operating system boot is finish.

Add a line with the *dv0_manager* and all desired command line parameters and FINISH THE LINE with the ampersand character.

```
/home/dv0/dv0_manager –force_date &
```

The inclusion of the ampersand character at the end of the line is mandatory or the Linux boot process never end. If you plan to launch *dv0_manager* in a separately script, it could be interesting to use its return value

3.2.1 *dv0_manager* return values

If there is something wrong, this program returns prematurely with the following codes:

- 255 (-1) if there is some syntax error at the command line
- 254 (-2) if the serial channel device selected is not free or no HeaderUSB is connected
- 253 (-3) if the socket library doesn't open the port 6900 or whatever declared with *–port* parameter. Try another port

Finally, restart Linux and execute the *dv0_test* program:

```
cd /home/dv0
./dv0_test
```

If every is fine, the program shows the message "Connection to localhost successful". Else, either there is a problem with the command line parameters, or with the serial channel or with the socket port.

To know what the problem is, begin with killing the running *dv0_manager* (if it still runs):

```
killall dv0_manager
```

Then execute *dv0_manager* direct from the shell who will tell which one of the three failure condition explained in 3.2.1 is responsible for the malfunction.

3.3 *dv0_manager* command line parameters

This section describes the command line parameters accepted by the dv0_manager, their meaning, limits and default values. The "default value" is the value that *dv0_manager* uses if the corresponding parameter is not present at the command line call.

Table 1 shows all valid parameters accepted by the dv0_manager.

Table 1. *dv0_manager* parameters summary

Parameter	Value	Default value
-serial_port	Port device	/dev/ttyAMA0
-port	Port	6900
-ip_address	IP Address	localhost
-login	Login name	none

3.3.1 Parameter *-serial_port <port>*

Port is the name of the device where the header board is attached. Default value is */dev/ttyAMA0*, intended for Raspberry Shields, but for the HeaderUSB must be an USB-Serial emulator like */dev/ttyUSB0* or */dev/ttyUSB1*, etc.

Connect the HeaderUSB to the USB port and list the */dev* directory

```
ls /dev/ttyU*
```

The response should be:

```
/dev/ttyUSB0
```

if the HeaderUSB is the only one serial emulator connected to the system.

Note: For the sake of clarity, all following examples omit the `-serial_port /dev/ttyUSB0` **parameter, but it is mandatory for a HeaderUSB board.**

Default value: */dev/ttyAMA0 (Raspberry Pi)*

3.3.2 Parameter *-port <port number>*

This is the TCP port number that dv0_manager listens to accept connections from *DV0 API* or *dv0_test*. It is not usual to change the default value of 6900 but in some cases, this value could be caught by other application.

Default value: 6900
Minimum value: 5000

3.3.3 Parameter *-ip_address <ip_address>*

This parameter is useful **only** when the user applications or the *dv0_test* run in other machine that the Linux system. The *<ipaddress>* must be equal to the interface IP where the connection to a local net is made. If all they run inside the same machine, the IP address is "localhost" which is the default value and so this parameter is not needed.

To know what IP address is assigned to an net port, execute the command:

```
ifconfig -a
```

For example, the result of this command can be:

```
eth0  Link encap:EthernetHWaddr b8:27:eb:c0:fb:76
      inet addr:10.0.0.2  Bcast:10.255.255.255
```

(…only two lines are shown…)

Then the call to *dv0_manager* should be

```
dv0_manager –ip_address 10.0.0.2
```

Default value: localhost

3.3.4 Parameter *-login <login name>*

When this parameter is set, all incoming connections to *dv0_manager* must be opened with the giving login name and with their corresponding password. For example, if the call to *dv0_manager* is

```
dv0_manager –login pi
```

then, the initial DV0 API function Open must be

```
Open("localhost", 0, "pi", "raspberry")
```

and the *dv0_test* should be

```
dv0_test –login pi –password raspberry
```

Note that *dv0_manager* needs to gain access to the Linux password files in order to check the validity of the password and that means that it must be runnning with root privileges. See 3.6.1 for more information

3.4 *dv0_test* command line parameters

The dv0_test program is intended to be a system tool for testing board connections, verifying *dv0_manager* correctness and programming Linux scripts.

This section describes the command line parameters accepted by the *dv0_test*, their meaning, limits and default values. Table 2 summarizes these parameters

Table 2. *dv0_test* parameters summary

Parameter	Value	Default value
-ip_address	IP Address	localhost
-port	Port number	6900
-login	Login name	None
-password	Password	None
-get_manifest	Address	None

3.4.1 Parameter *-ip_address <ip address>*

Use this parameter when *dv0_manager* has been called with this parameter as well. As explained in 3.3.3 section, if *dv0_manager* is invoked as

```
dv0_manager –ip_address 10.0.0.2
```

then, to communicate with *dv0_manager*, *dv0_test* must be called as

```
dv0_test –ip_address 10.0.0.2
```

NOTE: when both *dv0_manager* and *dv0_test* run in the same machine, they can use the interface “localhost” to establish the link, which is the default value for both of them. In this case, there is no need to set this parameter.

3.4.2 Parameter *-port <Port number>*

This parameter must be set only if the default port 6900 is occupied by another application forcing dv0_manager to be called with this parameter. For example:

```
dv0_manager –port 11200
```

then,

```
dv0_test –port 11200
```

3.4.3 Parameter *-login <login name>*

When *dv0_manager* has been called with this parameter then all further connections to its must supply not only the same login name but its password.

For example, as in the 3.3.4 section, when *dv0_manager* is called like

```
dv0_manager –login pi
```

then, dv0_test must supply the same login and its valid password

```
dv0_test –login pi –password raspberry
```

Note: This password is not encrypted while travelling through internet.

3.4.4 Parameter *-password <password>*

Used always together with *-login* parameter described above.

3.4.5 Parameter *-get_manifest <address>*

The manifest is a readable text who explains the features available from an DVO input/output board. The value of <address> must match with the micro-switch selector of the board that has to be tested.

For example, if there is a model IO_A board connected which micro-switch address selector is 7, then, when executing

```
dv0_test –get_manifest 7
```

the response will be

```
Board name    : IO_A
PWM Outputs   : 3
Pulse Inputs  : 3
Analog Inputs : 2
Digital output group number 0 has 2 outputs
Digital output group number 1 has 4 outputs
Digital input group number 0 has 3 inputs
```

Which are the contents of a model IO_A board.

It is worth to note that if this command is successful, that means that the whole installation is correct: *dv0_manager* is running, it is connected with the HeaderUSB control unit and the HeaderUSB DV0 Bus cabling is correct as well.

3.4.6 Return values of *dv0_test*

On successful, dv0_test returns 0. If not, it returns prematurely with the following return values

- 255 (-1) dv0_manager does not respond. Check IpAddress and Port
- 254 (-2) dv0_manager refuses login or password
- 253 (-3) DVO header board is not connected
- 252 (-5) Invalid board address
- 251 (-6) Parameter syntax error

3.5 *DV0* API

The DV0 API contains a set of functions intended to manage the HeaderUSB control unit and all DVO input/output expansion board connected to the bus. There are a free implementation of this API, for Java, C++ and Python languages.

WARNING:

This data sheet summarizes the features of DV0 API functions related to HeaderUSB board. It is not intended to be a comprehensive reference source. For more information, refer to the DV0 API Reference Guide for C++, Java and Python at www.devalirian.com, in the Technical Information section

The steps for managing the HeaderUSB control board programmatically are:

- Create an instance of the class DV0
- Call the function Open
- If successful, call HeaderUSB functions related.

In the following sections, an example of each language will be given creating instance, calling Open function and calling related functions and more complete examples can be found at the end of this section.

Table 5 summarize the functions related to HeaderUSB. Note that only the name of the function is listed in the table below. This is because the parameters and return values are slightly different among the three implementation (Java, C++ and Python).

Table 5. HeaderUSB related functions

Name	Description
`Open`	Initial and mandatory function to connect with dv0_manager
`GetHeaderInfo`	Obtains status and version info of the HeaderUSB control unit

3.5.1 Creating an instance of DV0 class

Java, C++ and Python are all object oriented languages and is useful to encapsulate all the accessing to the API throughout one object. All subsequent managing functions will pass through this variable. Let's assume that this variable will be called *dv0*. For C++ or Java, the creation of this variable is thoroughly:

```
DV0 dv0; // C++ or Java
```

And so is for Python, but slightly different (actually, it is not a class)

```
import dv0
```

3.5.2 Connecting to *dv0_manager*

The connection to *dv0_manager* is made through the function Open.

Function *Open* generic description
`int Open(IpAddress, Port, Login, Password);`
Open connection with the dv0_manager. Mandatory for all other members. Blocking.
Parameters:
IpAddress is the IP where dv0_manager listens to incoming connections. By default is "localhost" that suits perfectly is this application runs in the same machine that dv0_manager does. If a remote connection is desired, call dv0_manager with the argument -ip_address followed by the IP address of the internet interface of your board (use ifconfig to know it) and pass this value to the current IpAddress parameter. **Port** is the UDP port where dv0_manager listens to incoming connections. By default is 6900 but DV0_manager can listen to whatever port selected by command line argument -port. In this case, give the same value to current Port parameter. Useful only if another application had catch this port. **Login** and **Password** are required if the dv0_manager daemon is called with -login <username> argument. Current Login parameter must match with <username> and a valid Password must be entered
Return values
Returns 0 if connection is successful DV_NOT_CONNECTION if dv0_manager does not respond. Check IpAddress and Port DV_INVALID_LOGIN if dv0_manager refuses login or password
C++ Syntax
`int Open(char *IpAddress = NULL, int Port = 0, char *Login = NULL, char *Password = NULL);`
Java Syntax
`int Open(String IpAddress, int Port, String Login, String Password);`
Python Syntax
`def Open(ipAddress, port, login, password):`

Example: Connect to a *dv0_manager* running in the same Raspberry that the current program, using de default port and with no login required. The *dv0_manager* call can be:

```
/home/dv0/dv0_manager -serial_port /dev/ttyUSB0
```

From a C++ user program, then:

```
DV0 dv0;
…
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
…
} else DisplayError(r);
```

From a Java user program, then:

```
DV0 dv0;
…
int r = dv0.Open(””, 0, ””, ””)
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('',0,'','')
if (res == 0):
    print ("Connected to dv0_manager")
else:
```

```
    DisplayError(res)
```

Example: Connect to a *dv0_manager* running in a Raspberry connected to internet through the interface 10.0.0.2, and the current program running somewhere, login name 'pi' required (with default password 'raspberry') , using de default port. The *dv0_manager* call can be:

```
/home/dv0/dv0_manager -ip_address 10.0.0.2 -login user1 -serial_port /dev/ttyUSB0
```

From a C++ and Java user program, then (assuming that password of *user1* is "12345xx"

```
DV0 dv0;
…
int r = dv0.Open("10.0.0.2", 0, "user1", "12345xx")
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('10.0.0.2',0,' user1','12345xx')
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
```

3.5.3 API functions related to HeaderUSB

There is only one API function that can be used with the HeaderUSB: GetHeaderInfo

Function *GetHeaderInfo* description

```
int GetHeaderInfo (Info[8])
```

Fills the array Info with information from the HeaderUSB control unit.

Parameters:

```
The meaning of each occurrence of the array Info is
Info[0] = Header board version number: 3 for a HeaderUSB
Info[1] = Unused
Info[2] = Unused
Info[3] = Number of DV0 transmission errors detected
Info[4] = Number of DV0 Input/Output boards that have been detected
Info[5] = Percent of HeaderUSB control unit's memory used
Info[6] = Unused
Info[7] = Unused
```

Return values

```
Returns 0             if successful
DV_NOT_CONNECTED      if previous Open call had failed or connection has been canceled
```

C++ Syntax

```
int GetHeaderInfo (unsigned char Info[8]);
```

Java Syntax

```
int GetHeaderInfo (byte Info[8]);
```

Python Syntax

```
def GetHeaderInfo ():
# Returns a 2-elements tuple containing an array Info (as in C++ or java) and an integer
# (return value)
    return Date, Result
```

3.5.4 API examples

In this section, a complete example of how to manage HeaderUSB related functions are described both in C++ and Python languages. Java example is omitted because it is so close to C++ that only adds confusion. However, there is an example of how to program an Android APP with the DV0 API, that is not included in this section for the benefit of simplicity but it can be found at www.devalirian.com, inside the Technical Information section.

3.6.4.1. C++ ExampleHeaderUSB.cpp

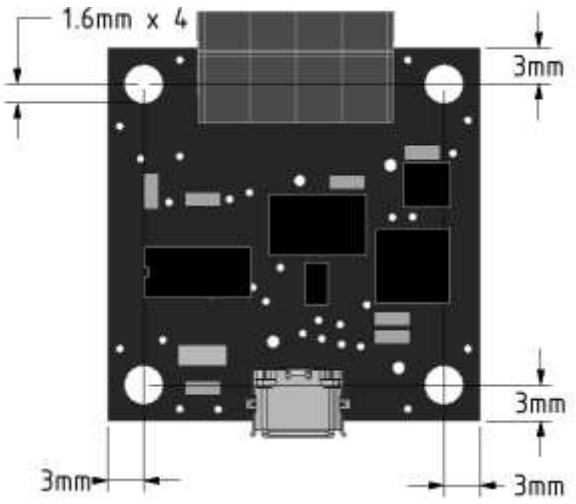
```
#include <iostream>
using namespace std;
#include "DV0.h"

char *ServAddress = NULL; char *LoginName = NULL; char *Password = NULL; int Port = 0;
DV0 dv0;

void DisplayError(int r);
int DoTest(void);
int SetDate();

int main() {
	int r;
	// Trying to connect
	r = dv0.Open(ServAddress, Port, LoginName, Password);
	if (r == 0) {
		cout << "Connection to dv0_manager successful " << endl;
		r = DoTest();
		if (r != 0) DisplayError(r);
	} else DisplayError(r);
	return r;
}

int DoTest(void) {
	int option, r, y, m, d, h, min, sec;
	// Print menu
	cout << "Menu:" << endl;
	cout << "1-GetHeaderInfo" << endl;
	// Get user option
	cin >> option;
	// And execute
	switch (option) {
		case 1:
			//---------------------------------------GetHeaderInfo
			unsigned char Info[8];
			r = dv0.GetHeaderInfo(Info);
			if (r == 0) {
				cout << "Version: " << (int)Info[0] << endl;
				cout << "CRC err: " << (int)Info[3] << endl;
				cout << "Boards : " << (int)Info[4] << endl;
				cout << "Mem %  : " << (int)Info[5] << endl;
			}
			return r;
	}
	cout << "Invalid option" << endl;
	return 0;
}

void DisplayError(int error) {
	switch (error) {
		case DV_NOT_CONNECTION:
			cout << "dv0_manager does not respond. Check IpAddress and Port" << endl;
			break;
		case DV_INVALID_LOGIN:
```

```
                        cout << "dv0_manager refuses login or password" << endl;
                        break;
                case DV_NOT_CONNECTED:
                        cout << "DVO header board is not connected" << endl;
                        break;
                case DV_INVALID_BOARD:
                        cout << "Invalid board address" << endl;
                        break;
                case DV_NOT_SUPPORTED:
                        cout << "Invalid peripheral" << endl;
                        break;
                default:
                        cout << "Unexpected error code" << endl;
                        break;
        }
}
```

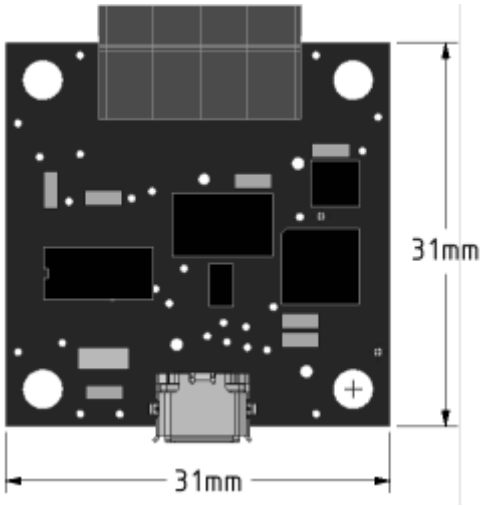
3.6.4.2. Python ExampleHeaderUSB.py

```
import sys
import dv0

def DisplayError(error):
    if error == 0:
        return
    if error == dv0.DV_NOT_CONNECTION:
        print ("dv0_manager does not respond. Check IpAddress and Port")
    elif error == dv0.DV_INVALID_LOGIN:
        print ("dv0_manager refuses login or password")
    elif error == dv0.DV_NOT_CONNECTED:
        print ("DVO header board is not connected")
    elif error == dv0.DV_INVALID_BOARD:
        print ("Invalid board address")
    elif error == dv0.DV_NOT_SUPPORTED:
        print ("Invalid peripheral")
    else:
        print ("Unexpected error code: ", error)

# Assuming that the Raspberry is not this computer and connected as 10.0.0.2
# Also, dv0_manager is called like "dv0_manager -ip_address 10.0.0.2 -login pi"
res = dv0.Open('10.0.0.2', 0, 'pi', 'raspberry')
# Assuming that the Raspberry is this computer
# and dv0_manager is called just like "dv0_manager"
# res = dv0.Open('',0,'','');

looping = True
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
    looping = False
while looping:
    print ("1-GetHeaderInfo")
    option = int(input("Enter option:"))
    if option == 6:
        # ----------------------------------------- Get HeaderUSB or HeaderUSB example
        Info, res = dv0.GetHeaderInfo()
        if (res == dv0.DV0_OK):
            print ("Version: %d", Info[0]);
            print ("CRC err: %d", Info[3]);
            print ("Boards : %d", Info[4]);
            print ("Mem %  : %d", Info[5]);
        else:
            DisplayError(res)
```

4. MECHANICAL DRAWINGS



Hole dimensions (top view)



Board dimensions (top view)

5. ACCESSORIES

The HeaderUSB control board comes with the following accessories:

- One DV0 socket. Model 20020004-D041B01LF

IMPORTANT NOTICE

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

deValirian MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.**

deValirian disclaims all liability arising from this information and its use. Use of ***deValirian*** devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless ***deValirian*** from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any ***deValirian*** intellectual property rights.

HeaderUSB board is not designed to be radiation tolerant

Please be sure to implement in your equipment using the safety measures to guard against the possibility of physical injury, file or any other damaged cause in event of the failure of HeaderUSB board. deValirian shall bear no responsibility whatsoever for you're your use of HeaderUSB board outside the prescribed scope or not in accordance with this manual.

Reproduction of significant portions of ***deValirian*** information in ***deValirian*** data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. ***deValirian*** is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Web Site: www.devalirian.com

Mail info: info@devalirian.com