IOPoint-USB 16 Isolated High Current Digital Outputs
Manual and Programmer's guide
2007-06-02A

GETTING STARTED

Contents of your package:

- IOPoint-USB PCB Assembly
- USB A to B Cable

To get started with your IOPoint-USB, you will need the driver and some example programs to try on your device.

Installing on Windows 2K or XP

Windows 2K and XP users should download the appropriate driver ZIP package from the product's webpage:

http://www.bibaja.com/products/?page=iopoint_usb

Extract this ZIP file to be used during installation.

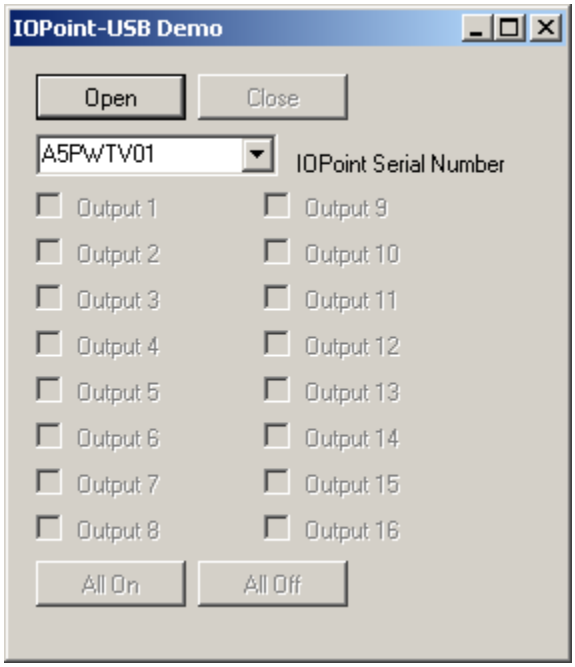
Now you are ready to install:

1) Plug USB A to B cable into computer and into IOPoint-USB
2) When prompted, select to install the driver yourself from a known location
3) Browse to the unpacked ZIP archive's directory and click OK
4) When installation finishes, restart computer

The IOPoint-USB is ready to go. The Visual Basic demo application provides a quick demonstration of how to access and use your IOPoint-USB programmatically. To use this application:

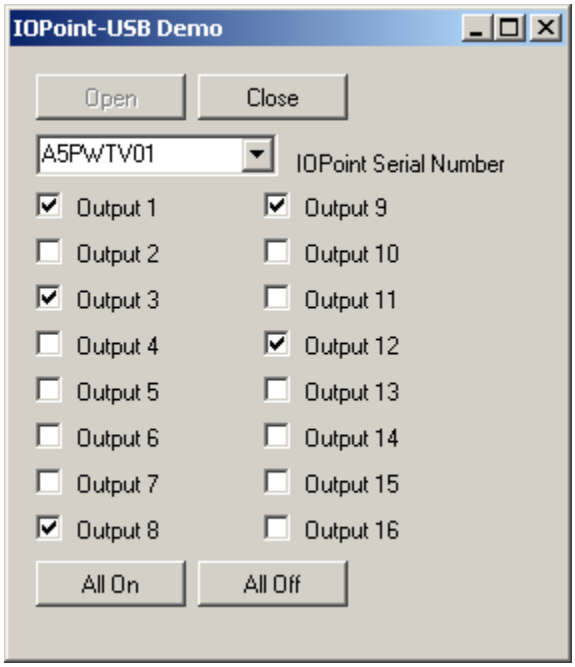
1) Download and extract the Visual Basic Demo App Installer ZIP archive from the web link above
2) Launch Setup.exe
3) The IOPoint-USB Demo will be installed and automatically launched

The IOPoint-USB Demo window should appear that looks like this:



The serial number of your IOPoint-USB should appear in the box beneath the "Open" button. If no device is found, make certain your IOPoint-USB is installed, attached, and the USB POWER LED is illuminated. Then close and restart the IOPoint-USB Demo.

Click "Open" to open your device. The Output checkboxes should all be enabled now allowing you to turn outputs on and off. The "All On" and "All Off" buttons at the bottom provide shortcuts to set all of the outputs to on or off. Here's how the app looks after opening and turning on a few outputs:



If you have multiple IOPoint-USB devices attached, you can select them from the list and click the Open button to open them. Changing the list entry will show you the outputs states for each opened IOPoint-USB. If an IOPoint-USB is closed, the outputs will be grayed out (disabled) but they will not be turned off.

Installing on Linux

Before installing the IOPoint-USB on a Linux machine, you must first install the following:

- libusb version 0.1.7 or later
- libftdi version 0.9 or later

Download libusb from:

http://libusb.sourceforge.net/

Download libftdi from:

http://www.intra2net.com/de/produkte/opensource/ftdi/

Once you have installed both of these you are ready to download the Linux demo application from Bibaja:

http://www.bibaja.com/products?page=iopoint_usb

Unpack the tar.gz archive, then run "make" to build the examples. If you are just building the command line tools, type "make command". If you are just interested in the TCL library, type "make tcllib". If you want to build both, type "make".

Now you can launch "./example.sh" to run an example of the command line tools, or you can launch "./example.tcl" to run an example of the TCL based library. Make sure you are running as root when you run these examples or you will not have the permissions to access /proc/bus/usb. You may also discover the device node for the IOPoint-USB under /proc/bus/usb and change the permissions to world read/write.

There is also a way to configure the scripts under /etc/hotplug to automatically set the permissions when a new IOPoint-USB is plugged in. To create a script, you will need to know the IOPoint-USB's VID/PID are 0x0403/0xE339.

CONNECTING THE HARDWARE

Now that you've installed all the software, you're ready to connect the hardware. For this you will need the following:

- A 12V power source with enough current to power your target device
- Quick disconnect female crimp connectors
- The device you wish to turn on

For the automotive environment, you will be using the 12V power of the car. Sum up the total current requirement for all outputs and install a fuse between the car's 12V power and the two 12V inputs of the IOPoint-USB. Be certain to use sufficient gauge wire if you plan to use the entire 144 amp driving capacity of the IOPoint-USB device. That's 9 amps per output, 16 outputs, or 16 * 9 = 144 amps.

For other environments, you may be using a 12V power supply. As with automotive applications, make certain you have sufficient gauge wire going to one or both of the 12V power supply connections to support your application's maximum current requirement.

The GND connection on the IOPoint-USB does not carry much current, so a small wire from this terminal to the car's chassis ground or ground return of your power supply is sufficient. This connection is used to bias the gates of the MOSFET switches allowing them to turn on.

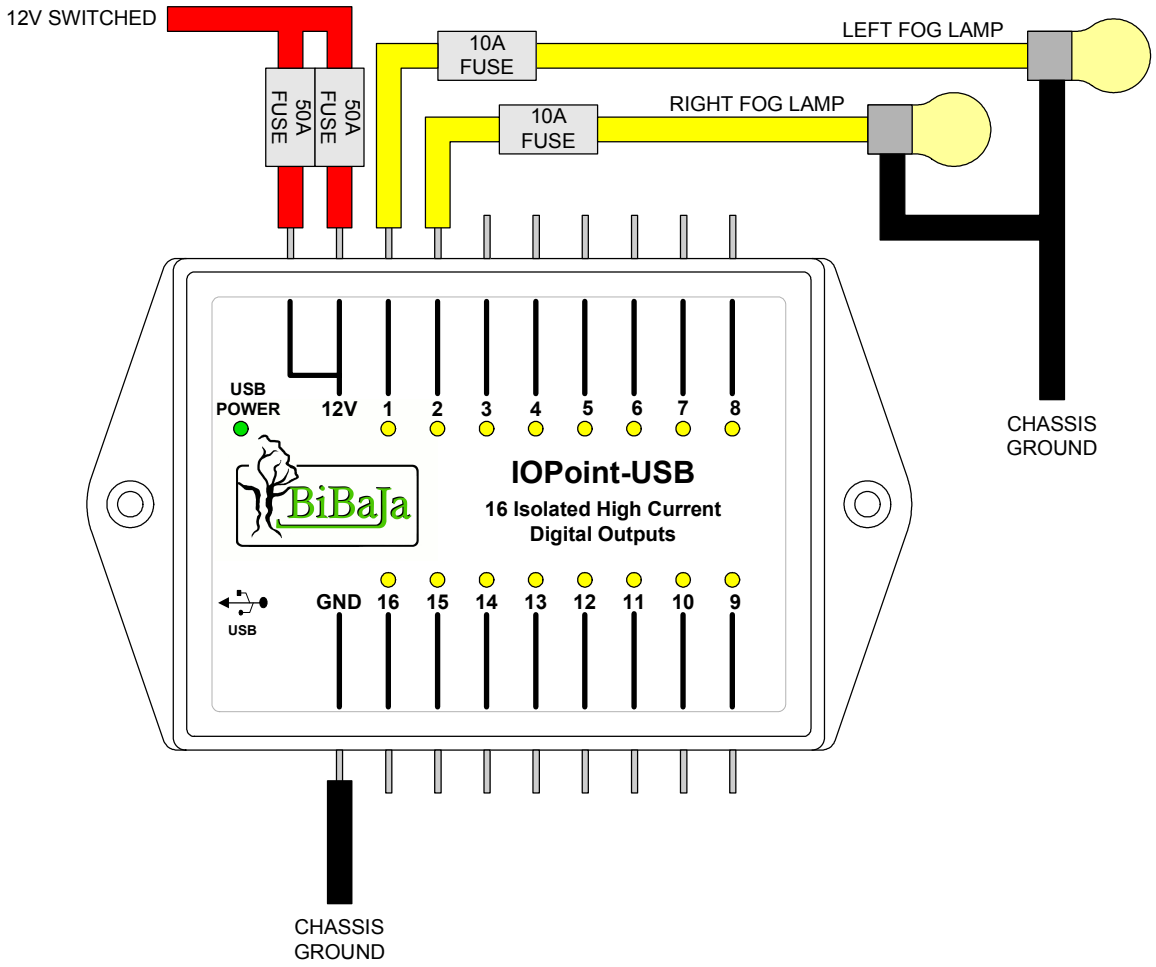
Make certain your 12V and GND connections are solid, and make sure they are not reversed.

WARNING: Reversing the power supply connections causes the protection diodes in the IOPoint-USB to conduct. If your power supply does not have a current limit, you must install a fuse to protect against accidental reverse of the power supply to prevent damage to the protection diodes. Check your power connections carefully.

To test at this step, plug your IOPoint-USB into your computer, turn on your 12V power source, and launch the demo application. Using this application, you should be able to turn on outputs and see the LED illuminate for that output. This indicates you have 12V connected properly.

Now you are ready to attach your devices. Let's assume this is an automotive application and we are controlling two fog lamps mounted towards the front of the car. We will use outputs 1 and 2 to control the left and right fog lamps independently. Connect a wire from output 1 and route it through a 10 amp fuse to the left fog lamp. Connect the other side of the fog lamp to the nearest chassis ground on the car. Repeat this for the right fog lamp.

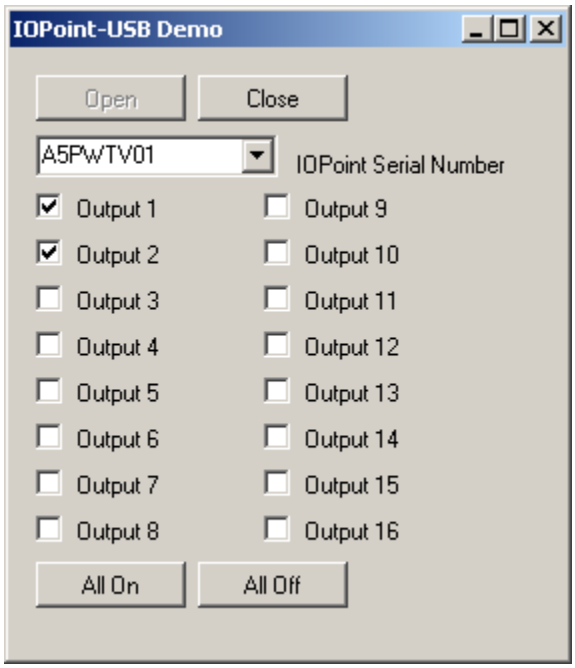
When you are finished, your connections may look like this:



Note in this drawing we used two 50 amp fuses instead of one 100 amp fuse. Choose whatever makes your wiring simple and reliable. If you need 50 amps or less, then connecting one terminal is sufficient.

Note the IOPoint-USB does have built-in 10 amp PTC fuses for sustained overload protection. These may not protect from a dead short on an output, such as the fog lamp yellow wire shorting to the chassis. A short on this wire would allow greater than 55 amps of current to be drawn through the output in 300us time before the PTC reacts and opens the circuit. If there is any chance such a short could occur, you may wish to use external fast blow fuses to prevent possible damage to the mosfet switch in the IOPoint-PL.

Now that your connections are made, power up the demo application and turn on outputs 1 and 2:



Now your fog lamps should both be on. If they are not, confirm the LEDs are illuminated on your IOPoint-USB, check your connections, check your fuses, and check to make sure the fog lamps are not burned out.

To test using the Linux command line tools with a single device installed:

```
iop_set_output 1 1
iop_set_output 2 1
```

Or, if you have read ahead and know how to make raw bytes to send to the device:

```
iop_write 1 3
```

Using the TCL examples on either Linux or Windows:

```
load libiopusbtcl.so ;# On Linux
load iopusb_tcl.dll  ;# On Windows

set iop [ iop_open ] ;# Open the first device found
iop_write $iop 1 3   ;# Set outputs 1 and 2
iop_close $iop       ;# Close the device
```

PROGRAMMER'S GUIDE

So now you want to write your own program. Whether you program with VB, C, or are a TCL scripting fanatic, we've got you covered. If you are interested in a JNI for Java, we can make that too.

In this section we'll go into the low level stuff and explain how the device works and tell how to talk to the device using the FTD2XX Direct Drivers from FTDI.

What Makes it Tick?

What makes the IOPoint-USB tick is a small chip from FTDI known as the FT245R. You can read more about that chip on FTDI's website and download the FTD2XX Programmer's Guide while you are there. You will need this guide to understand how to talk to your IOPoint-USB directly:

http://www.ftdichip.com/

The FT245R is a USB FIFO chip. What does this mean? Simply put, it means you squirt bytes in from the USB side and they pop out a parallel FIFO on the other side of the FT245R. Using a little bit of glue logic, ok, some glue logic and isolation barriers and mosfet magic, we take those parallel bytes and pop them out of the FIFO and use the information to set the state of the 16 high current output drivers.

Controlling the Outputs

Now you know we have an FT245R chip in there pushing out 8 bits at a time to our interface circuitry. The next piece you need to understand is how to format those 8 bits to control individual outputs.

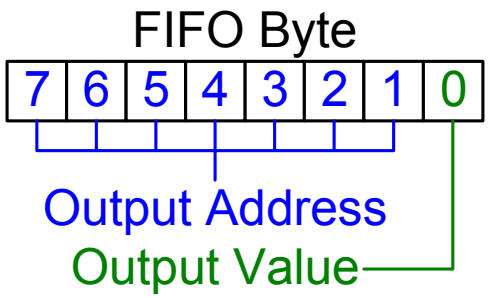
The 8 bits are divided into two groups: Output Address and Output Value.

Bits are ordered from 7 down to 0, with 7 being the most significant bit and 0 being the least significant.

Bits 7 to 1 are the Output Address. This means the value encoded in these bits selects the output to control. For devices like this, we operate using a zero base. This means OUT1 is Output Address 0, and OUT16 is Output Address 15. Put another way, the Output Address is the OUT number minus 1.

Bit 0, the only bit left, is the Output Value. 0 means off, and 1 means on.

Putting it together:



Now let's say you wanted to turn on outputs 1, 8, and 16. How do you convert these into raw FIFO Byte values to turn on the outputs? Here's an equation to help:

FIFO Byte = (Output Address * 2) + Output Value

Remember also:

Output Address = Output Number – 1

So for outputs 1, 8, and 16:

FIFO Byte Out 1 = (0 * 2) + 1 = 1
FIFO Byte Out 8 = (7 * 2) + 1 = 15
FIFO Byte Out 16= (15 * 2) + 1 = 31

Now if you use the FTD2XX driver API and open the device, then write the values 1, 15, and 31 to the device, outputs 1, 8, and 16 will be on. To turn them off again, write 0, 14, and 30 (Subtract the 1 used to turn them on).

Here's a quick reference table for the output values:

Output	State	Hex Value	Decimal Value
1	OFF	0x00	0
2	OFF	0x02	2
3	OFF	0x04	4
4	OFF	0x06	6
5	OFF	0x08	8
6	OFF	0x0A	10
7	OFF	0x0C	12
8	OFF	0x0E	14
9	OFF	0x10	16
10	OFF	0x12	18
11	OFF	0x14	20
12	OFF	0x16	22
13	OFF	0x18	24
14	OFF	0x1A	26
15	OFF	0x1C	28
16	OFF	0x1E	30
1	ON	0x01	1
2	ON	0x03	3
3	ON	0x05	5
4	ON	0x07	7
5	ON	0x09	9
6	ON	0x0B	11
7	ON	0x0D	13
8	ON	0x0F	15
9	ON	0x11	17
10	ON	0x13	19
11	ON	0x15	21
12	ON	0x17	23
13	ON	0x19	25
14	ON	0x1B	27
15	ON	0x1D	29
16	ON	0x1F	31

IOPoint-USB Device Identification

Bibaja does not have it's own USB Vendor ID, so we enrolled in FTDI's vendor supplied PID program. FTDI allocated a block of PIDs from their address space to us to use for our FTDI based devices.

For this product, you will need to know that the VID and PID are:

VID: 0x0403
PID: 0xE339

Why is this important? To identify our device on the USB bus. When you use FTDI's FTD2XX driver, it will poke around on the USB bus and return a list of all FTDI devices attached. This means you may find other manufacturer's products as well.

In our example source, we use FTDI's FT_GetDeviceInfoList() API call from the FTD2XX driver to retrieve a list of these devices and compare them to a known ID. In this case, we use the ID 0x0403E339. In case you cannot possibly imagine where that number came from, it is the VID/PID smashed together in one 32-bit integer in hexadecimal representation.

See the TCL example source available from our website for the "iop_list" TCL procedure source in C. This demonstrates how to use the FTD2XX driver to probe the bus and locate the attached IOPoint-USB devices.

Putting it All Together

Now you know where to find the programmer's guide (FTDI's website). You know to use the FTD2XX direct driver API. You know the FIFO byte format to turn outputs on and off. Now what?

Well, the basic flow goes like this:

1) Examine our example source, or use the demo app as is. Or send us suggestions for better demo apps.
2) Write a program to scan the USB bus using code like our example application to find the IOPoint-USB
3) Call the FT_OpenEx() routine to open a specific Serial Number. This is printed on the label on your unit. You can always skip step 2 if you will never have more than 1 unit, or if you wish to always manually enter the serial numbers.
4) Call the FT_Write() routine to write FIFO bytes to turn outputs on and off
5) Call the FT_Close() routine to close the connection to the IOPoint-USB and exit. Make sure you leave the outputs in the state you want before closing your program. A safe thing to do might be to dump a bunch of even numbers using FT_Write() to turn off all outputs before calling FT_Close()

That's basically it. Download and use the example source from our website. These examples will help you use the FTD2XX driver to implement your own application, or

you may be able to leverage these examples directly to create your application and save you some time.