Control Devices (GPIOs and beyond)

How to send any data from Blynk apps to hardware and use it to control a device

Getting Started -> Control Devices with Blynk

How Data Gets From Blynk To The Device

Blynk can control any supported device remotely using Blynk.Console web interface or Blynk.Apps for iOS and Android.

When you tap a button or swipe the slider in the mobile app, the value is sent to the device through a Datastream using Blynk protocol.

Datastream is a channel that tells Blynk what type of data is flowing through it.

For example, you can tell the button to set the GPIO on your device HIGH or LOW or you can send a specific value based on the state of the button.

You can control the GPIOs directly by using Digital and Analog Datatreams, but we highly recommend using Virtual Pin Datastreams

Virtual Pin Datastream

Virtual Pins are designed to exchange **any data** between your hardware and Blynk. Anything you connect to your hardware will be able to talk to Blynk. With Virtual Pins you can send something from the App, process it on the microcontroller, and then send it back to the smartphone. You can trigger functions, read I2C devices, convert values, control servo and DC motors, etc.

Virtual Pins can be used to interface with external libraries (Servo, LCD, and others) and implement custom functionality.

Why Use Virtual Pins To Send Data To Device?

- Virtual pins are hardware-independent. This means that it's far easier to port your code from one hardware platform to another in the future (when you realize that the NodeMCU is far better than the

Arduino Uno + ESP-01 that you started with, for example).

- You have far more control over what your widgets do when using Virtual Pins. For example, if you want a single app button to switch multiple relays on or off at the same time then that's simple with virtual pins, but almost impossible using digital pins.
- Virtual pins are more predictable (stable if you like) than manipulating digital pins.

How do Virtual Pins **relate to the GPIO pins on my hardware?**

Virtual Pins are really just a way of sending a message from the app to the code that's running on your board (via the Blynk server).
There is no correlation between Virtual Pins and any of the physical GPIO pins on your hardware. If you want a Virtual Pin to change the state of one of your physical pins then you have to write the code to make this happen.

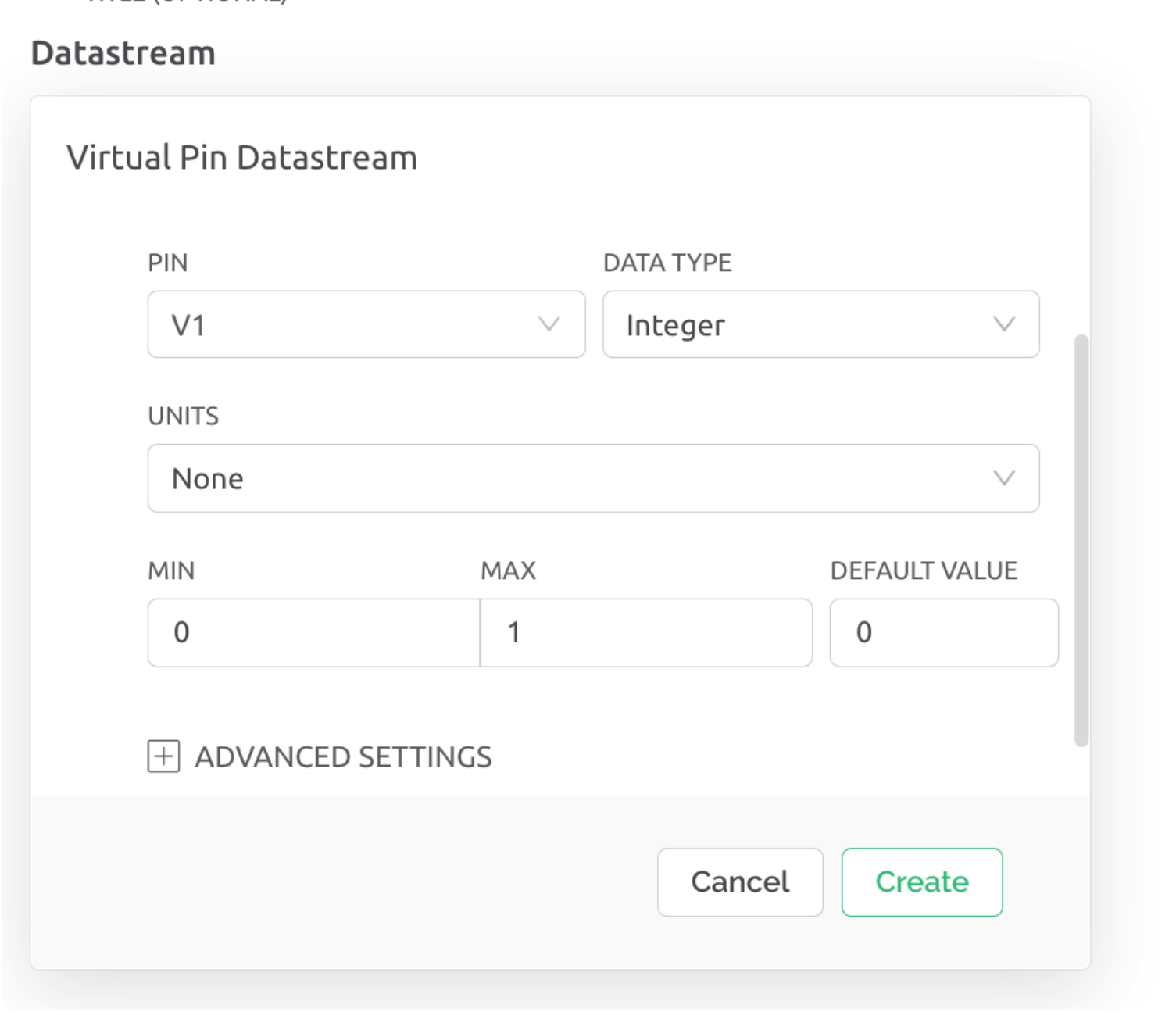
Basic Principles Of Sending Data With Virtual Pin Datastream

We'll use an example of a Power Switch to remotely turn the device on and off with Blynk.Console web interface.

1. Create a new Template or go to existing one Blynk.Console ->Templates
2. Go to Template -> Dashboard - add Switch Widget
3. Open widget settings - Create Datastream - Virtual Pin

Switch Settings

Set to Integer data type, connected to Virtual Pin 0 (V0). In Blynk.Console we'll leave the values set to 0 and 1, so the widget sends a 0 when it's turned off, and a 1 when it's turned on - like this:



Now the widget is ready to send 0/1 through the Virtual Pin Datastream V1. Click **Save and Apply** to save the template and apply changes.

Refer to these articles if needed:

- **How to create a device from Template******
- **Quick Template setup******

The BLYNK_WRITE(vPin) function

Now every time you press the switch, 0 or 1 will be sent to your hardware. Let's prepare the code to capture these values when they come.

In your C++ sketch, you can add a special function that is triggered automatically whenever the server tells your device that the value of your virtual pin has changed. \

This special function is called `BLYNK_WRITE`. Think of it as meaning that the Blynk.Cloud is telling your hardware “I'm WRITING something to Virtual Pin V1”.

So, for Virtual Pin 0, your sketch would need this bit of code adding…

```
BLYNK_WRITE(V1)
{
  // any code you place here will execute when the virtual pin value changes
  Serial.print("Blynk.Cloud is writing something to V1");
}
```

That’s okay if you want the same code to execute regardless of whether the button widget was turned on or off, but that’s not much use in real life, so we need to find out what the new value of the virtual pin is. Don’t forget, this will be a `0` if the button widget is off, and a `1` of it’s on.

Obtaining values from the virtual pin

The server sends the current Virtual Pin `value` to the hardware as a parameter, and this can be obtained from within the `BLYNK_WRITE(vPin)` function with a piece of code like this…

```
int virtual_pin_value = param.asInt();
```

This tells the code to get the value parameter from the virtual pin and store it in the local integer variable called `virtual_pin_value`.
We can then use this value to do different actions if the button widget is now on, compared to if it is now off.

Some datastreams send their values at text, or double/float point numbers (integers are whole numbers, double point variables are one with numbers to the right of the decimal point).
To allow you to use these values the Blynk library also allows the following:

```
param.asStrng()
param.asFloat()
```

But, as we are dealing with just zeros and ones, which are integers, we’ll use the `param.asInt` method.

How to control the physical GPIO pins on my board?

If you're familiar with C++/Arduino programming you'll probably know about `pinMode` and `digitalWrite` commands already. These are what we use to control the physical pins of your device from within the `BLYNK_WRITE(vPin)` function.
For those people who aren't familiar with these commands, I'll give a brief summary here - but feel free to learn more by searching the internet.

The pinMode command tells your board how a particular pin is going to be used. The three most common commands are:

```
pinMode(pin, OUTPUT);
pinMode(pin, INPUT);
pinMode(pin, INPUT_PULLUP);
```

You only need to issue a pinMode command once, when your device boots up, so this command goes in your `void setup()`

The digitalWrite(pin, value) command sets the specified pins LOW (zero volts) or HIGH (3.3v or 5v, depending on your type of board). This is how you energise your relay, LED etc.

Bringing it all together…

This example code assumes that we want to control digital pin 2 on your board…

Note that you can only have one `void setup` in your sketch, so the pinMode statement needs to be copied into your existing `void setup`.

```
void setup()
{
  pinMode(2, OUTPUT); // Initialise digital pin 2 as an output pin
}
```

That's it really, you now have the same functionality from your virtual pin as you would have had if you'd used a digital pin.

I'm now going to cover some extra stuff - some of which is specific to working with virtual pins, but some also apply if you use digital pins instead…

```
BLYNK_WRITE(V0) // Executes when the value of virtual pin 0 changes
{
  if(param.asInt() == 1)
  {
    // execute this code if the switch widget is now ON
    digitalWrite(2,HIGH);  // Set digital pin 2 HIGH
  }
  else
  {
    // execute this code if the switch widget is now OFF
    digitalWrite(2,LOW);  // Set digital pin 2 LOW
  }
}
```

Helpful tips when working with Virtual Pins

Dealing with Active LOW devices

Many devices, such as the onboard LED attached to GPIO2 of the NodeMCU and many relay boards, are energized by a LOW signal rather than a HIGH signal. This means that, if you use the code example above, the LED/Relay will be off when the switch widget shows On, and vice-versa.

There are several ways around this issue. You could change your switch widget output settings to be 1/0 rather than 0/1 - so that an On state send a "0" value to the app like this…

Syncing the output state with the app at startup

When the device starts up (reboots) it won't be aware of the state of the button widget in the app, so may be in a different state than what the button widget shows in the app. This will be rectified when you toggle the button widget in the app, but that's not a very satisfactory solution.

We can force the Blynk server to send the latest value for the virtual pin, by using the `Blynk.syncVirtual(vPin)` command. This causes the corresponding `BLYNK_WRITE(vPin)` command to execute.

To automatically run the `BlynkSyncVirtual(vPin)` command when the device connects to the Blynk server (which will normally be after a reboot but could also be following a disconnection) we can use another special function called BLYNK_CONNECTED, like this…

```
BLYNK_CONNECTED()
{
```

```
  Blynk.syncVirtual(V0);  // will cause BLYNK_WRITE(V0) to be executed
}
```

Pin numbering

If you're working with an Arduino then it's all quite simple. General Purpose Input/Output (GPIO) pin 2 is labeled "2" or "D2".
However, if you're using a NodeMCU type of device then the manufacturers decided to make things a little more complicated. The numbers screen printed onto the board are not the GPIO numbers. You have to translate these NodeMCU "D" numbers onto actual GPIOs as follows…

`Label GPIO`
`D1 5`
`D2 4`
`D3 0`
`D4 2`
`D5 14`
`D6 12`
`D7 13`
`D8 15`

So, in the example above where we controlled GPIO2 with the command `digitalWrite(2,HIGH)` it's actually the pin labelled "D4" on the NodeMCU that we're turning on and off.

The Arduino IDE does allow you to use the NodeMCU's "D" pin numbers directly rather than GPIOs, like this:

```
digitalWrite(D4,HIGH);
```

But this approach makes it much more difficult to use your code on different types of devices if you ever need to.

Some NodeMCU physical pins need to be avoided

Some of the pins on the NodeMCU aren't really suitable for connecting some types of devices to. In particular, if GPIO0 (the pin labeled D3) is pulled LOW at startup then the device won't execute the sketch but will enter programming mode, waiting for a new sketch to be uploaded instead. There's more info on this topic: ESP8266 GPIO pins info, restrictions and features FAQ

How to trigger multiple actions (e.g. turn 4 relays on/off) with a single button in the app?

You said *"if you want a single app button to switch multiple relays on or off at the same time then that's simple with virtual pins" but how do we do that*? - It's really very simple with Virtual Pins.

Let's say that you have four relays that are all controlled by four different button widgets attached to virtual pins (V1 to V4). These allow independent control of each of the relays, but you then want another button widget, which we'll attach to virtual pin 5, that can turn all of the relays on or off with just one click. When this button turns on/off all of the relays it also needs to update the 4 button widgets (attached to V1 to V4), so that they are also all on or off.

Here's what the `BLYNK_WRITE(V5)` function would look like to do this...

```
BLYNK_WRITE(V5) // Executes when the value of virtual pin 5 changes
{
  if(param.asInt() == 1)
  {
    // execute this code if the switch widget is now ON
    digitalWrite(2,HIGH);  // Set digital pin 2 HIGH
    digitalWrite(4,HIGH);  // Set digital pin 4 HIGH
    digitalWrite(5,HIGH);  // Set digital pin 5 HIGH
    digitalWrite(12,HIGH);  // Set digital pin 12 HIGH

    Blynk.virtualWrite(V1,1);  // Turn the widget attached to V1 On
    Blynk.virtualWrite(V2,1);  // Turn the widget attached to V2 On
    Blynk.virtualWrite(V3,1);  // Turn the widget attached to V3 On
    Blynk.virtualWrite(V4,1);  // Turn the widget attached to V4 On
  }
  else
  {
    // execute this code if the switch widget is now OFF
    digitalWrite(2,LOW);  // Set digital pin 2 LOW
    digitalWrite(4,LOW);  // Set digital pin 4 LOW
    digitalWrite(5,LOW);  // Set digital pin 5 LOW
    digitalWrite(12,LOW);  // Set digital pin 12 LOW

    Blynk.virtualWrite(V1,0);  // Turn the widget attached to V1 Off
    Blynk.virtualWrite(V2,0);  // Turn the widget attached to V2 Off
    Blynk.virtualWrite(V3,0);  // Turn the widget attached to V3 Off
    Blynk.virtualWrite(V4,0);  // Turn the widget attached to V4 Off
  }
}
```

As you can see, instead of just doing a digitalWrite to one GPIO pin, we're doing 4 pins one after another. `Blynk.virtualWrite` commands are then issued, which will update the button widgets to match the digital pins.

Getting Started - Previous
Send Data From Hardware To Blynk

Next - Getting Started
Events

Last modified 2mo ago

WAS THIS PAGE HELPFUL?