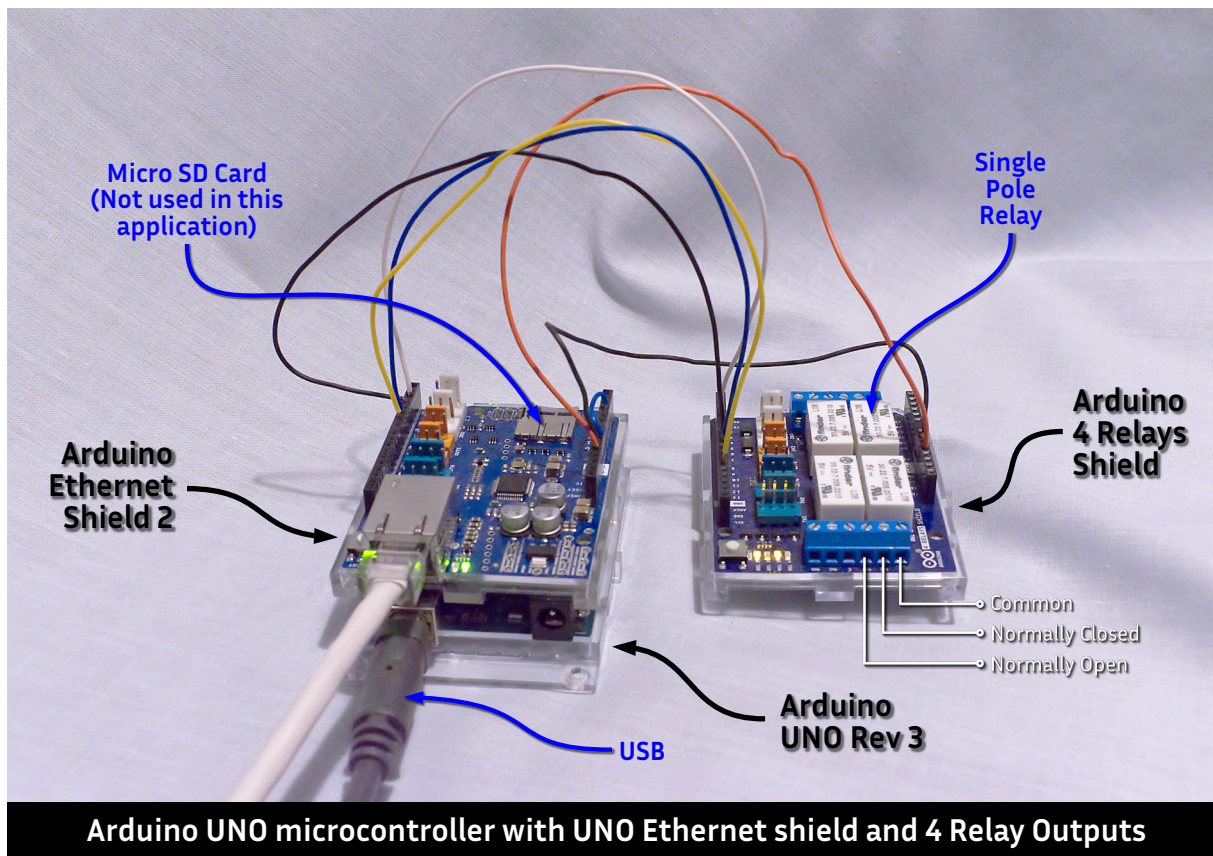


Low cost

General Purpose Output (GPO)

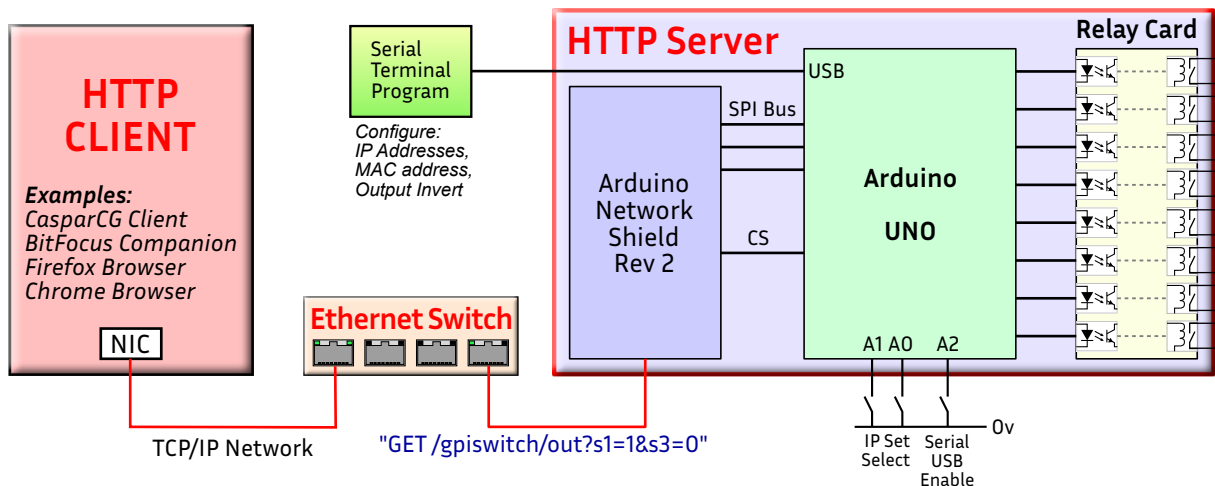
Using HTTP GET Control Messages

This note describes a simple network-connected GPO (General Purpose Output) interface with eight outputs implemented using a low-cost Arduino UNO microcontroller and off-the-shelf relay interface board or boards. The state of each GPO output is controlled by HTTP GET messages.



The prototype shown above was used during development and testing of the GPO interface described in this note. The relay board includes small LED indicators showing which relays are activated. The indicators are to the right of the push-button switch, and to the left of the blue screw terminals block.

The interface output states are volatile, initialised to OFF when the unit is powered or reset. The system block diagram is below.



The diagram shows an example of the GET control message structure used by the application. The message contains the target address with output 1 switched on and output 3 switched off.

`GET /gpswitch/out?s1&s3=0`

A web browser can issue the command using an address entry similar to:

`http://192.168.42.191:2000/gpswitch/out?s1&s3=0`

where the unit is configured to listen for traffic on address 192.168.42.191 port 2000.

Assembling an instance of the interface requires some simple soldering to add the wiring between the Arduino unit and any relay interface board or LED diode, and simple mechanical skills to mount the hardware in a suitable container. An external 5V DC power supply is required. This must output sufficient current to operate both the microcontroller and the relay interface card. Some relay units may be powered via the USB connection.

Some GPO interface instances are part of permanent infrastructure and only require their IP address set at installation. Other interface instances are bookable resources deployed wherever temporary extra outputs are required. This style of GPO application may need a different address entering into the interface.

This interface uses the USB serial connection from the Arduino to set the IP address properties, storing the edited values in EEPROM inside the microcontroller. DHCP address allocation is not supported by the interface.

The GPO interface supports deployment on multiple used networks by storing four sets of IP address values in on-board EEPROM memory. Two single-pole switches select the active TCP/IP address set when the interface boots or is reset. A single pole switch enables the serial connection to a terminal program used to set the IP addresses, MAC address and output state invert. Serial connection is undesirable unless configuration is in progress. The serial connection should be disabled in deployed units.

The application returns a stringified JSON data item that reports the current state of the outputs when the switch states are set, or when no commands are present on the address.

The following examples show the status poll and the response:

Read status: **http://192.168.42.191:2000/**

Read status: **http://192.168.42.191:2000/gpiswitch/out**

Response: **{"s1":1,"s2":0,"s3":0,"s4":1,"s5":0,"s6":0,"s7":0,"s8":0}**

The Arduino sketch (program) can be used with a UNO Rev 3 microcontroller coupled with an Arduino Ethernet Shield Rev 2. It should also be possible to run the code in a NANO unit, but this **MUST** use a WizNet ethernet interface because ENC28J60 shields often used with a NANO processor require too much of the available program and variable memory to allow the core application to be present.

The interface provides galvanic isolation by use of :

1. The transformer built into the ethernet interface (protects the network infrastructure).
2. Opto-isolators in the input stage of a relay output card protect the microcontroller from output equipment faults.
3. Relays providing opening or closing output signal connection. The relay outputs are typically connected to low-voltages on the external equipment input stages, but some relay modules **are** capable of mains load switching. Use appropriate installation mechanisms to minimise risk of mains voltage exposure to those configuring and maintaining the GPO unit.

The Arduino sketch (program) should be downloaded into the host hardware **before** the network card and relays are connected to the host.

The user must decide if they want to connect direct to the relay contacts, typically through a screw terminal block on the relay card, or provide connection to the relay outputs via standard connectors such as XLR or audio jacks.

Up to eight relay outputs are available per interface assembly when using a UNO microcontroller.

Larger numbers of outputs can be implemented by using a larger microcontroller such as the Arduino Mega. The program sketch needs editing to support the extra outputs. Increases in buffer sizes for ethernet traffic and serial configuration are needed, as are larger look-up tables that map between logical and physical switch outputs.

Key Features

The GPO interface uses a generalised design supporting a wide range of deployed control operations.

- It uses off-the-shelf hardware for the microcontroller and ethernet interface. The program has been tested with Arduino UNO microcontroller fitted with an Arduino Ethernet Rev 2 shield.
- Off-the-shelf relay boards with opto-isolated control inputs can be controlled by the program. The specific relay board in use defines the maximum switched current and voltage.
- The software always supports 8 outputs from the microcontroller. The number of output relays can be any number between 1 and 8 depending on the relay interface card(s) fitted. The unit also supports low-current LED emitters without using relay switches.
- The interface software is available under ISC open source licence. This allows end users to freely adapt the code base to support more outputs via a larger microcontroller such as the Arduino Mega.
- Customised networking properties for the module are stored in EEPROM within the Arduino. These properties are user configured via a serial port terminal connection to the interface system.
- The unit stores 4 sets of TCP/IP network address properties. This allows the interface unit to quickly set network properties when the module is moved between commonly used areas, each area having it's own network address range.
- There is a user programmed output invert property. This allows a logical 'ON' switch request to output either a low voltage or a high voltage as required by the specific relay interface fitted. The invert setting is per output, enabling relay modules to be mixed as needed.
- The interface uses static outputs enabling control of rehearse/transmission style light indicators. Pulsed GPO operation is implemented by the control client sending time offset 'on' and 'off' messages.
- The interface is simple to power using a USB plug adapter, if the relay switching current is low, or using a brick PSU.

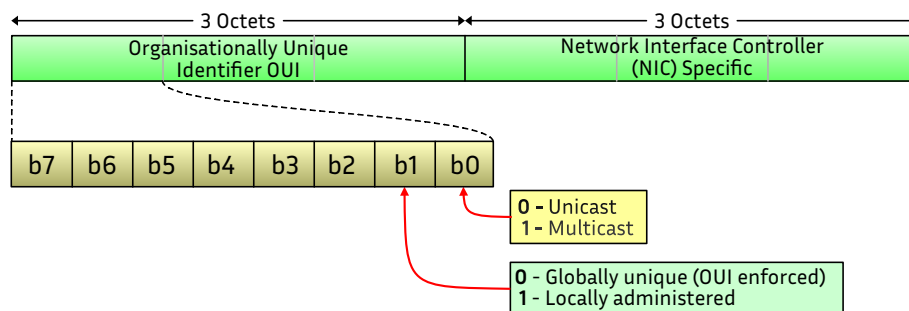
Arduino GPO Output Pin Allocation

The interface software is designed to work on an Arduino UNO microcontroller with an Arduino Ethernet shield. This shield uses WizNet 5100 or WizNet 5500 chip set and commonly includes a micro-SD card holder on the ethernet module. The SD card enable/disable is controlled by Arduino pin 4, so this pin cannot be used for GPO control. The controlled output pin use is shown in the table below:

| GPO No | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---------|----|----|----|----|----|----|----|----|
| UNO Pin | D2 | D3 | D5 | D6 | D7 | D8 | D9 | A3 |

Ethernet MAC Number

Every ethernet interface requires a 48-bit Media Access Control (MAC) number that is unique in the network segment on which the interface is used. This number may be globally allocated and stored in the interface firmware, or locally administered. The address type is signalled in the lowest two bits of the first address byte.



Modern practice operates the six octets as two groups each of three octets. The first three identify the manufacturer of the NIC, and the second three are a unique value allocated by the ethernet interface manufacturer.

The ethernet NIC in an Arduino installation requires a MAC address, but most hardware does not have a unique value allocated during manufacture. An exception is a modern ethernet UNO/MEGA shield from the official Arduino shop which have a sticker showing the globally allocated value for that interface. The value is not hard-coded into the interface and must be passed to the hardware as part of the application program setup() function operations. Most example Arduino sketches show a default value of 0xDE 0xAD 0xBE 0xEF 0xFE 0xED (DEAD BEEF FEED).

This GPO interface does not require an edited Sketch with different MAC for each deployed system. Instead it uses the serial terminal capability within the Arduino to allow the MAC value to be entered and stored in EEPROM within the Arduino microcontroller. It is still important to use a unique MAC value with all the networks where the device is deployed.

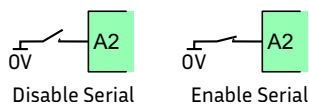
If the ethernet module has a sticker with a globally administered address use this value in the GPO configuration. Otherwise, use a MAC address that starts 0xDE (defined as unicast locally administered) plus 5 bytes chosen by the user.

Where multiple HTTP GET GPO units may be deployed on a network keep a list of the addresses allocated to ensure unique allocation.

Serial Link Control

The Arduino board includes a USB connection used for downloading the Sketch (program) and transferring serial data between a host computer terminal program and the Arduino.

The HTTP GET GPO application uses the serial link to allow the MAC address, relay output state invert and IP properties to be edited and stored in the internal EEPROM. In normal GPO operation the serial link is unused and disabled.

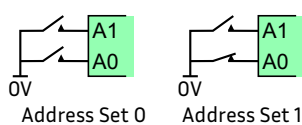


When the application boots it sets Arduino pin A2 (Analogue 2) to operate as a digital input with an internal input pull-up resistor. Once the mode has been set, the pin input level is tested. If the pin is open circuit it reports a HIGH input, and the serial link is not enabled. If the pin is connected to earth the input reports LOW during the boot process and the serial link is enabled. The serial properties are:

9600 baud, 8 data bits, no parity, single stop bit

The low-level is only required at unit boot/reset because the pin state is only read during the boot process, so a momentary action switch or shorting link can be used when the serial link is needed.

IP Address Sets



The HTTP GET GPO unit supports four sets of IP addresses stored in the internal EEPROM of the Arduino. The multiple address are useful where a unit is frequently moved between two or three networks. The active address set is controlled via the state of pins A1 and A0 which are read when the OSC GPO boots. The pins are set as inputs with internal pull-up resistors and operate with inverted sense, so with the two pins open circuit the unit uses address set 0.

| Switch 1 | Switch 2 | Address Set |
|----------|----------|-------------|
| Open | Open | 0 |
| Open | Closed | 1 |
| Closed | Open | 2 |
| Closed | Closed | 3 |

Each address set contains 5 elements:

| Element No | Property Name | Example |
|------------|--------------------|----------------|
| 1 | IP Address | 192.168.42.190 |
| 2 | Subnet Mask | 255.255.255.0 |
| 3 | Gateway Address | 192.168.42.1 |
| 4 | DNS Address | 192.168.42.1 |
| 5 | Listen Port Number | 2000 |

Although the application uses a static IP address and does not require packet routing it may need to set a non-default subnet mask, for example 255.255.240.0. The ethernet library call that sets the subnet mask also requires the gateway address and dns address passed to the function. If there is not a gateway or DNS server on the network set the DNS and gateway to the first address in the subnet range. For example if using an IP address of 10.206.84.191 and a subnet mask of 255.255.240.0 set the DNS and gateway address to 10.206.80.1

Entering Address Data

The MAC address, output invert flags and IP address sets are all edited then stored into EEPROM using commands entered through the Arduino USB serial communication port. Hence the HTTP GET GPO unit **must** be connected to a host computer running a terminal program. Example terminal programs include:

- Arduino Development software **Serial Monitor** window (Windows and Mac OS),
- **Putty** on a Windows OS host (download from <https://putty.org/>),
- **SerialTools** on MacOS (free from the Apple Application store).
- **Putty** for Mac OS (available from several web sites)

If the Arduino development environment is available, this is the simplest terminal to configure and use. If the Serial Monitor is not visible in the Arduino development software enable it using the **Tools... Serial Monitor** menu.

Whichever terminal software is used the data link properties must be set as 9600 baud, 8 data bits, no parity with one stop bit.

The HTTP GET GPO serial input processing uses line feed as the end-of-line delimiter. The text parser ignores carriage return characters, hence the terminal program can be set to output line feed or carriage return plus line feed when the return key is pressed.

An advantage of the Arduino development tool is the **Tools Port:** menu lists available serial ports. The Serial Monitor window shows if the currently selected port is communicating with an Arduino, providing a simple comms check.

Windows OS host computers normally have port names that start with **com**, for example **com5**. Mac OS host communications ports are all members of the /dev root folder, hence their name starts /dev. The full name is often a long string such as **/dev/cu.usbmodem14401**.

The edit process can only modify the *active* IP address set. Select the HTTP GET GPO address set for edit using the switches or links on the unit. Reset or power the unit to activate the address set selection. Also ensure the serial link control is set to the position that enables serial communications (data pin A2 connected to 0V).

If the serial communications are enabled the terminal displays information about the current address set. The display format is illustrated below.

```
HTTP GET GPO Interface
Using IP Address set 0
Invert control 00000000
MAC DE:AC:AD:03:00:01
IP 192.168.42.191 port 2000
Mask 255.255.255.0
GW 192.168.42.1
DNS 192.168.42.1
Server has been started on port 2000
```

Unused EEPROM cells have all bits at one, an octet value of FF hexadecimal. Hence a new Arduino board that has only had the operating program loaded, or an IP address set that has not yet been configured, shows the value FF in all octets of the address listings.

Check the host to HTTP GET GPO serial connection by typing the command **show** and pressing the return key. The HTTP GET GPO sends a report similar to the startup display, but with a different first line display, no indication which set number is being edited and no information about the status modes.

```
Edited Properties
Invert control 00000000
MAC DE:AC:AD:03:00:01
IP 192.168.42.191 port 2000
Mask 255.255.255.0
GW 192.168.42.1
DNS 192.168.42.1
```

The same format of data display is output after each valid data entry. It is assumed the person editing the content is knowledgeable about TCP/IP address usage. Some input validation is applied, but the limited amount of memory in the Arduino means extensive checking is not possible.

If the entered information has detectably wrong syntax or an unusable value the current edit properties are not shown, just a single line that states **Error**.

The edited values are only stored in the EEPROM when the user inputs the **save** command, and the new values are activated when the HTTP GET GPO unit is reset or re-powered.

All edit commands are entered using **lower case**, with a single space character between elements of the command. The formats of the valid edit commands are listed below.

show

Display the contents of the edit stores.

set mac DE:AC:AD:01:00:01

Defines the MAC number to be used by the ethernet interface. The validation tests check that the address is 17 characters long, all address bytes are hexadecimal, and there is a colon separator between each octet value. The MAC address is common to all address sets.

set invert 00001001

Defines the processing actions in the GPO output. The final element of the command must contain eight characters, all with a value of 0 or 1.

When a bit is set at 1 the associated output level is inverted relative to the remote control request state. This invert capability supports relay boards that use active low interfaces. The leftmost digit in the string controls GPO 8, the rightmost controls GPO 1.

set ip 192.168.42.42

Sets the IP address for the HTTP GET GPO device to the address provided in the command. The input value is checked and an error reported if the value is not a valid IP4 address.

set port 2001

Sets the IP port number of the interface. HTTP GET clients send packets to this port number. The provided value is checked as a valid unsigned number on the range 0 to 65535.

set mask 255.255.255.0

Sets the network mask.

set dns 192.168.42.1

Sets the address for the DNS service. Because the HTTP GET GPO uses static addresses the DNS address is not used, but it should be set to a standard default value. Set the DNS address as the first allowed address within the address block defined by the network mask.

set gw 192.168.42.1

Set the gateway address used for packet routing. Usually set as the same address used for the DNS. The ethernet library has to receive a value so that it can set the network mask property to a non-default value (one other than 255.255.255.0).

set statusport 1234

Set the port number that will receive control and status request messages.

save

Saves the edited address set into the EEPROM store. The store mechanism checks the existing value in the EEPROM and only writes changed values. This minimises erase cycle wear in the hardware, although the EEPROM in the Arduino should support at least 100,000 erase cycles.

After the values are saved, **reboot** the HTTP GET GPO unit and check the addresses reported at power up. Finally disable the serial port use and reset the HTTP GET GPO unit.

Switch Control Message Format

The interface uses an http GET message to control the output switch states. The primary URL comprises the protocol descriptor, target IP address and port, the page address and the switch control element.

`http://192.168.42.191:2000/gpiswitch/out?s3=1&s6=0`

protocol IP address port page address switching command

If the switching command is omitted the message operates as a status request. To check the switch status using a browser enter an address in the format:

`http://192.168.42.191:2000/gpiswitch/out`

The switching command element starts with a question mark which is followed by one or more switch output settings. Each setting is a switch identifier, such as **s3**, a separator, **=**, and finally the output level, **0** or **1**. A single element is switched by a GET command:

`/gpiswitch/out?s3=1`

To use a browser to control the switch enter an address line with the format:

`http://192.168.42.191:2000/gpiswitch/out?s3=1`

Multiple switch elements can be set in a single GET operation by appending switch level set commands separated by an ampersand (**&**). To set switch 1 on, switch 3 off and switch 5 on the switching command element is:

`?s1=1&s3=0&s5=1`

All switching commands return the new status for **all** output switches. The status report format uses stringified JSON representation:

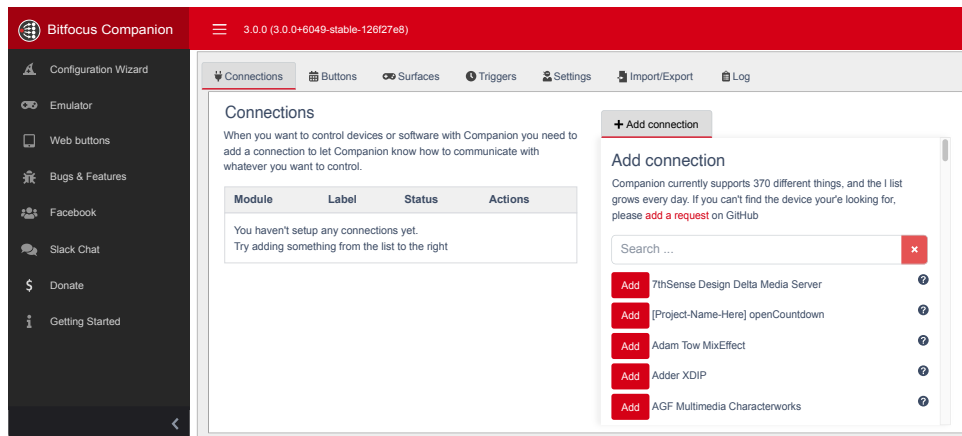
`{"s1":1,"s2":0,"s3":0,"s4":1,"s5":1,"s6":0,"s7":1,"s8":0}`

Control GPO outputs using BitFocus Companion

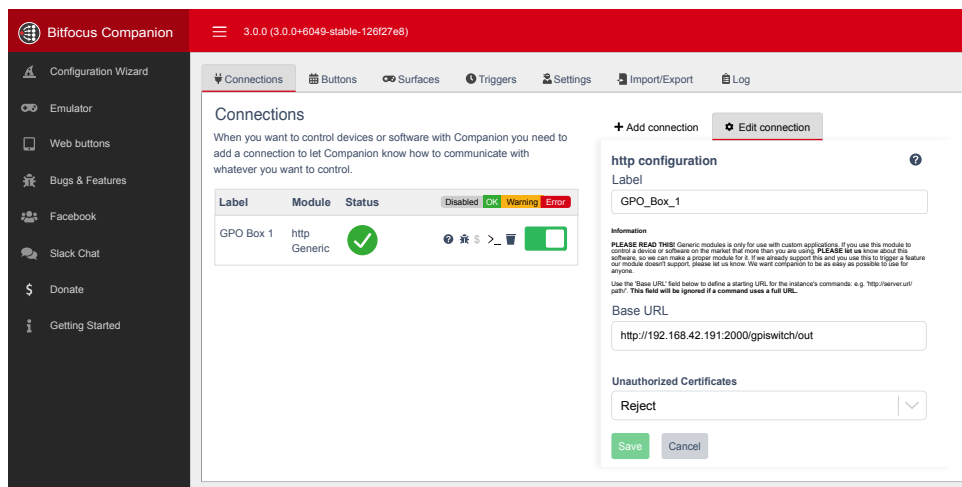
The generic http protocol available in BitFocus Companion can be used to control switches in the GPO unit. Companion version 3 or later is recommended because this can process the status report from the GPO interface. Version 2 can send the GET request, but does not process the response.

Assume there is an http GPO unit at IP address 192.168.42.191 using port 2000.

Run Companion, open the Companion GUI and select the **Connections** tab.



Type **generic http** into the Add connection tab search box. A single entry is displayed. Click the Add button. The display changes to the **Edit Connection** tab.



Enter a recognisable identifier into the http configuration **Label** field. This label will be used in the list of objects when editing a button control assignment. Note the text in the label field **must not** contain any space characters - use either underscore, hyphen, or camel case to show parts of the name. Examples:

GPO_Box_1

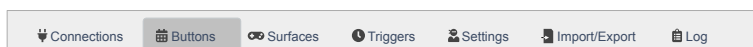
GPO-Box-1

GpoBox1

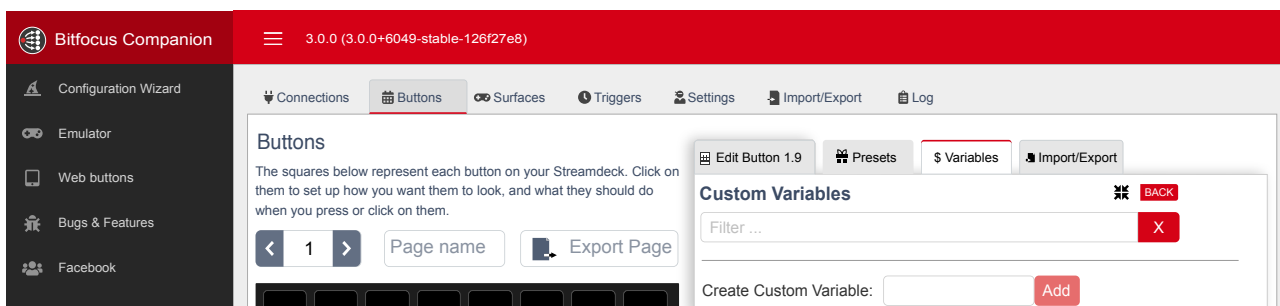
Enter the full URL address for the HTTP GPO device with port number and the switch control page address (/gpiswitch/out). Set the Unauthorized Certificates to **Reject**, then click the **Save** button.

Companion supports variables associated with device connections as well as user-defined **custom variables**. A custom variable can receive the response from the GPO device, extracting the status of a single switch (requires Companion version 3 or later).

User variables are configured via a form associated with the **Buttons** master tab.

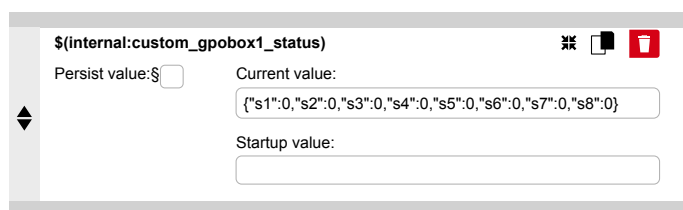


Select the Buttons tab, then select **\$ variables** which is in the third tab in the right-hand side of the form. The Custom Variables form, with no user variables yet defined, is illustrated below.

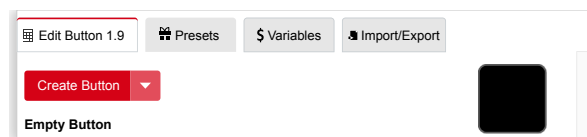


Enter a name, with no spaces, in the Create Custom Variables box. When a valid name is entered the background colour of the Add button (at the right of the entry field) becomes more saturated. The user entered name is combined with fixed data to create the name used in other entry and reporting boxes. The user-entered value of `gpobox1_status` becomes:

`$(internal:custom_gpobox1_status)`

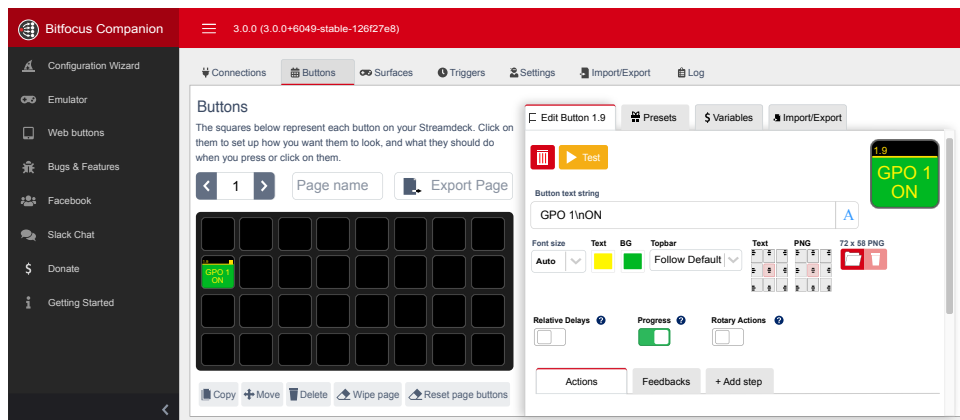


Once a value is assigned to the variable, that value is shown in the Current Value box as illustrated at the left.

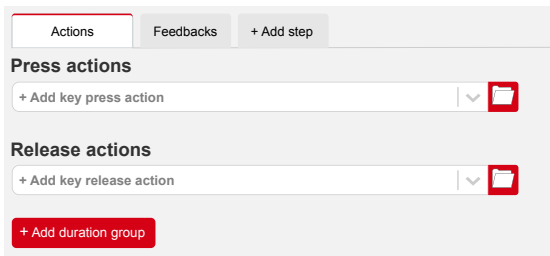


Select a Stream Deck button to configure using the keypad graphic. If the button actions are not yet defined a **Create Button** form is displayed within the Edit Button form zone (see left).

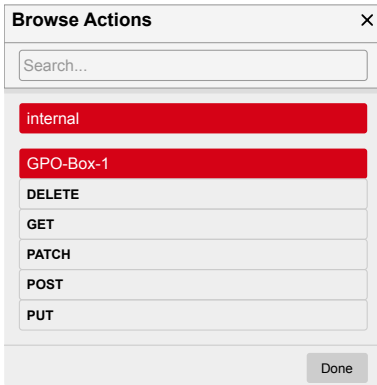
Click the “Create Button” box to set up a standard user-button entry form. Add text to display on the button, setting text and background colours as required. The `\n` in the button text string below starts a new line of text on the button display.



Scroll the button edit form to show the Actions entry section. The blank form is illustrated below.



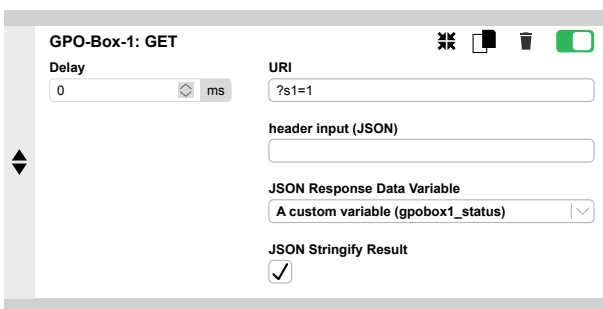
Click on the white folder icon in the red box at the right hand end of the Press Actions or Release Actions. This click displays a pop-up window that lists the available connections.



Select the connection named earlier in the configuration, GPO-Box-1 in this example.

Selecting the name displays the five supported actions for an http connection. Click the **GET** line once to add a single instance of a get request, then click the **DONE** button to close the list.

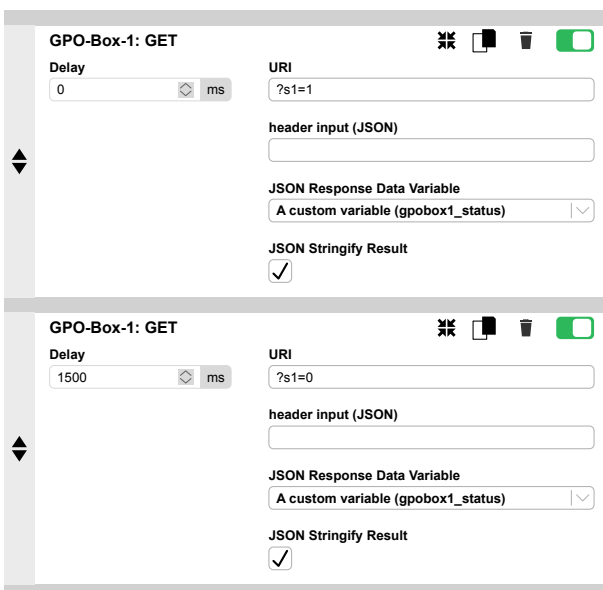
Note: multiple clicks on the GET add multiple action events to the button.



Edit the GET properties form. Specify the switching command in the URI box. The example at the left switches output 1 on.

If multiple outputs are to be switched the URI has several tokens each token separated by an ampersand.

Leave the header input box blank. Use the dropdown selector in the JSON Response Data Variable box to select the custom variable created earlier in the configuration process. That variable is used in the feedback programming of the button.



Generating a pulsed output uses two GET definitions in the button press actions.

Set the switch action of the second entry to the opposite of that used in the first entry. For example

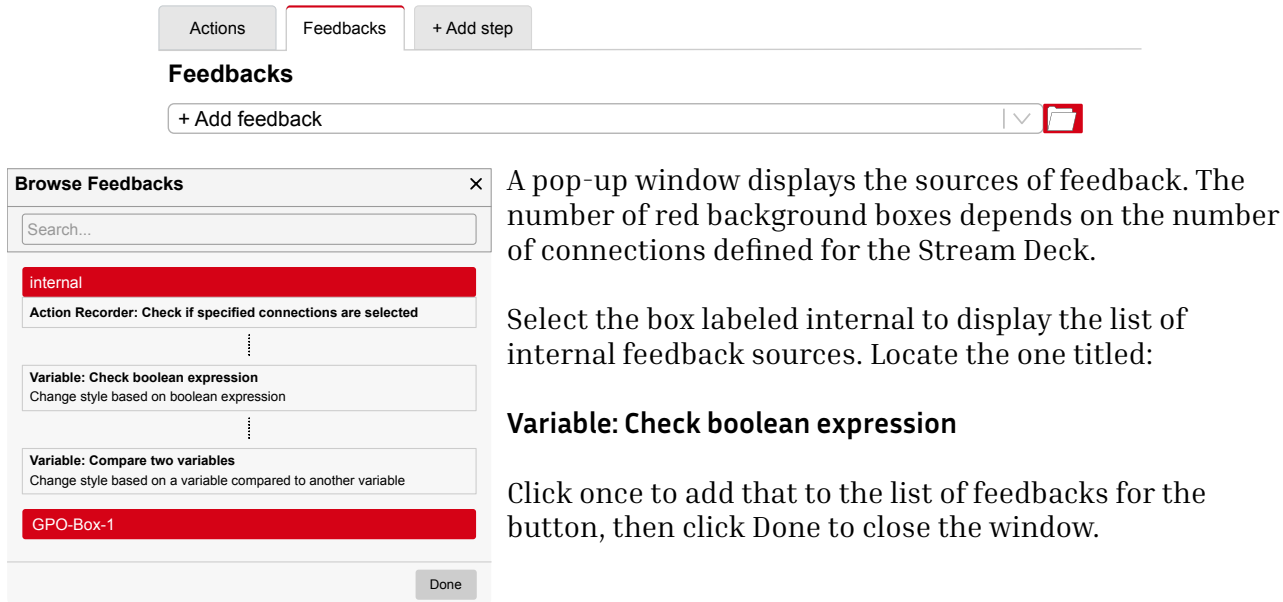
Entry 1 ?s1=1

Entry 2 ?s1=0

Set the required pulse duration as the delay value in the second definition.

The JSON string emitted by the GPO unit sets the value of the selected custom variable. The information in that variable can be used to create feedback on the Stream Deck button, for example setting the background colour of a button red when output 1 is switched on.

Switch to the Feedbacks tab. Click the folder icon in the red box to show the available sources of information.



Feedbacks

+ Add feedback

Browse Feedbacks

Search...

internal

Action Recorder: Check if specified connections are selected

Variable: Check boolean expression
Change style based on boolean expression

Variable: Compare two variables
Change style based on a variable compared to another variable

GPO-Box-1

Done

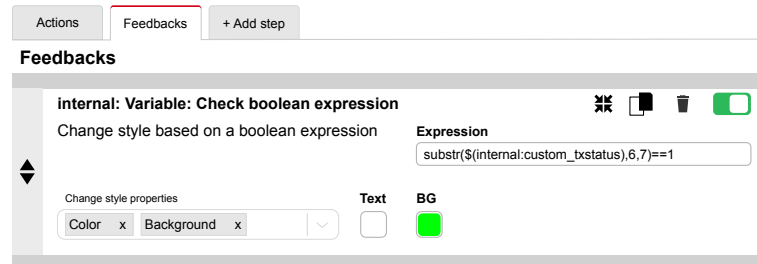
A pop-up window displays the sources of feedback. The number of red background boxes depends on the number of connections defined for the Stream Deck.

Select the box labeled internal to display the list of internal feedback sources. Locate the one titled:

Variable: Check boolean expression

Click once to add that to the list of feedbacks for the button, then click Done to close the window.

The feedback processing for the boolean expression is entered into the box labelled Expression, as illustrated below. The string in the example below is looking at the status of output 1.



Feedbacks

internal: Variable: Check boolean expression

Change style based on a boolean expression

Expression

substr(\$(internal:custom_txstatus),6,7)==1

Change style properties

Color x Background x

Text BG

The **substr(src, n, m)** function extracts a part of the **src** string provided as parameter 1, starting at character number **n** and including characters up to but not including character number **m**.

The stringified JSON string contains sequences of ASCII characters identifying the current output state of the switches. A possible output for the first three switches is illustrated below.

| Position | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | | |
|-----------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|-----|---|
| Character | { | " | s | 1 | " | : | 1 | , | " | s | 2 | " | : | 0 | , | " | s | 3 | " | : | 1 | , | ... | } |

Switch 1 status Switch 2 status Switch 3 status

The numbering of characters in a string starts at zero. In the above table we see that the output state of switch 1 is at character position 6, the status for switch 2 is at character position 13, and the status of switch 3 is at position 20. Hence to extract that single character for switch 1 we ask the substring function to start at character 6 and stop at character 7. The extracted character is tested for a match to a value by the **==** operator. The equivalence (**==**) test works, but the test should strictly be implemented as:

substr(\$(internal:custom_txstatus),6,7)=="1"

The start and end positions for all eight switches are shown in the following table.

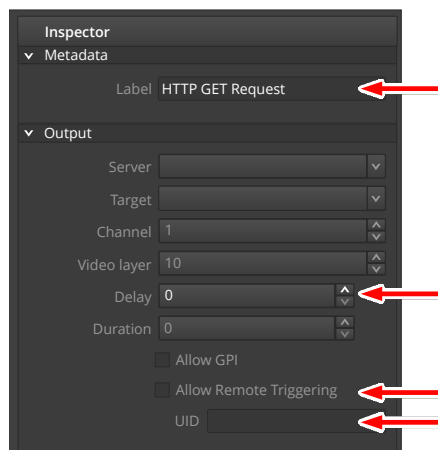
| Switch # | Start # | End # |
|----------|---------|-------|
| 1 | 6 | 7 |
| 2 | 13 | 14 |
| 3 | 20 | 21 |
| 4 | 27 | 28 |
| 5 | 34 | 35 |
| 6 | 41 | 42 |
| 7 | 48 | 49 |
| 8 | 55 | 56 |

If the comparison test “fails” the button colours remain at the defaults set for the specific button position.

Control GPO outputs using SVT CasparCG Client

The SVT CasparCG standard client has tools to output http GET and http POST messages. The GET tool is can be used to control the GPO interface. The CasparCG client receives the returned status JSON string which is recorded in the client log, but is not otherwise processed.

Add an http GET control to the rundown. This control is available in the Tools... Other library listing, or using a mouse right-click and selecting the Tools... Other context menu. The widget display in the rundown shows minimal labelling.

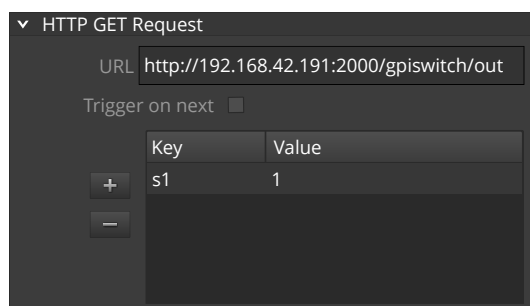


The control identification text defaults to “HTTP GET Request” but this can be updated by editing the Label field in the Metadata section of the Item Inspector.

Set the text to a phrase that identifies the control function, for example “Enable Mixer DSK”.

The Inspector Output property section has controls that set the post-trigger action delay, and enable remote control via OSC message. Enable remote control by ticking the **Allow Remote Triggering** box and entering the OSC address identifier in the UID box.

Assume there is a GPO box with IP address 192.168.42.191 listening on port 2000.



The switching data is entered into the HTTP GET Request properties section of the item Inspector. Enter the full address including the protocol, address, port and page address in the URL field.

The switch states to assert are entered as Key and Value elements. The example at the left shows the settings for switch 1 being set to 1. A blank key/value line is added by a click on the + button.

HTTP GET Request

URL:

Trigger on next ☐

| Key | Value |
|-----|-------|
| s1 | 1 |
| s3 | 0 |
| s5 | 1 |

The example to the left shows switching commands for switch numbers 1, 3 and 5.

When a pulsed GPO output is required use two HTTP GET requests in a group as illustrated below.

Rundown 01 x

LED Light flash via GPO #1

Duration: UID:

| | | | |
|----------|------------------|-------------|------|
| http GET | Switch light on | Delay: 0 | UID: |
| http GET | Switch light off | Delay: 2500 | UID: |

Set the first output item output delay to zero, and the second item output delay to define the pulse duration. Edit the Metadata Label fields to describe the function of each element.

Sources of Arduino and Relay Modules

The popularity of Arduino for many projects has produced a wide range of hardware suppliers. The open source nature of the Arduino board design means that there are also low-cost clones. Some suppliers offer screw terminal adapters for the Arduino, simplifying inter-module wiring.

RS Components - [web link](#)

The RS stocked ranges include UNO and NANO microcontrollers manufactured under the official Arduino label. RS also sell the official 4-channel relay shield and the WizNet Ethernet shield for UNO and MEGA boards. The ethernet module can piggy-back onto the controller card, but the relay unit requires wiring because some relay control pins are used by the ethernet interface.

CPC/Farnell - [web link](#)

CPC sell a wide range of Arduino and Arduino clone modules.

Arduino Store via Amazon.co.uk - [web link](#)

The full range of Arduino microcontrollers and shields are available from this web store.

AZDelivery via Amazon.co.uk - [web link](#)

This web store offers low-cost clones for many of the Arduino microcontrollers. They also have multi-unit bundles offering lower cost per unit than single purchase prices. The store sells terminal adapter boards (screwdriver connection module) for the NANO processor range, and an ENC28J60 series ethernet module for the NANO processors. Most microcontrollers are delivered with a USB A to Arduino USB cable.

AZDelivery also stock a range of relay interface modules, including one, two, four and eight relay per card modules. These require soldered connections to their controller board.

Elegoo via Amazon.co.uk- [web link](#)

Sell Arduino clones and small relay cards.

Appendix A - EEPROM Storage allocations

The memory address blocks used to store the configuration data are shown in the table below.

| Start | End | Information Stored |
|-------|------|--|
| 0x00 | 0x05 | MAC address |
| 0x10 | 0x17 | Output invert controls. 0=no invert, 1=invert. |
| 0x20 | 0x33 | User IP address set 0 |
| 0x40 | 0x53 | User IP address set 1 |
| 0x60 | 0x73 | User IP address set 2 |
| 0x80 | 0x93 | User IP address set 3 |

Within each address set memory block the relative offsets for the fields are shown in the following table.

| Offset | Content |
|--------|-----------------|
| 0x00 | IP Address |
| 0x04 | DNS address |
| 0x08 | Subnet mask |
| 0x0C | Gateway address |
| 0x10 | Port number |

| | |
|------------------------|------------------------------|
| Document written by | Andy Woodhouse |
| Document version | 1.00 |
| Document Date | 15 th August 2023 |
| Copyright | Andy Woodhouse |
| Arduino Sketch Name | HTTP_GET_GPO.ino |
| Arduino Sketch Version | 1.00 |