

Содержание

HTTP Communications	2
Processing Feedback	6
BC1 IR Input	12
Create a toggle from two discrete commands	14
Debug Monitor	15
Factory Default Procedure BC1 and BC2	16
Factory Default Procedure BC4	17
BC4 Master Reset Procedure	18
Update Internal Web Pages	21
Firmware Recovery BC4	24
GUI Channel Macro Tool	29
Integrating IP Cameras and Servers	31
Match Triggers BC1 and BC2	33

HTTP Communications

This article discusses methods and limitations for communicating with devices via HTTP.

Background

HTTP - ([HyperText Transfer Protocol](#)) is the underlying protocol used on the World Wide Web and is also often used to communicate with devices or services in the Custom Electronics space. HTTP most commonly uses TCP for the transport layer but is not the same as a native [TCP socket](#). For this reason, a [Socket Device](#) will not work for http communications. You must use a [Script Device](#) and select API for the protocol type. Creating the http strings requires an intermediate level of JavaScript knowledge. The next steps will demonstrate an example of communicating with a BC4 via http. While this may not be a practical application, the strings are simple and easy to test if you have a BC4 available. The http string format is different for BC1/2 controllers.

The BC4 does not run JavaScript so the example below will only run app side. This means http commands cannot be used in macros on a BC4.

HTTP Command Strings

The first critical step in creating the http driver is understanding the structure of the http strings.

To activate a relay on a BC4 with an IP address of 10.10.1.64, you could send the following string (you can test this from any browser):

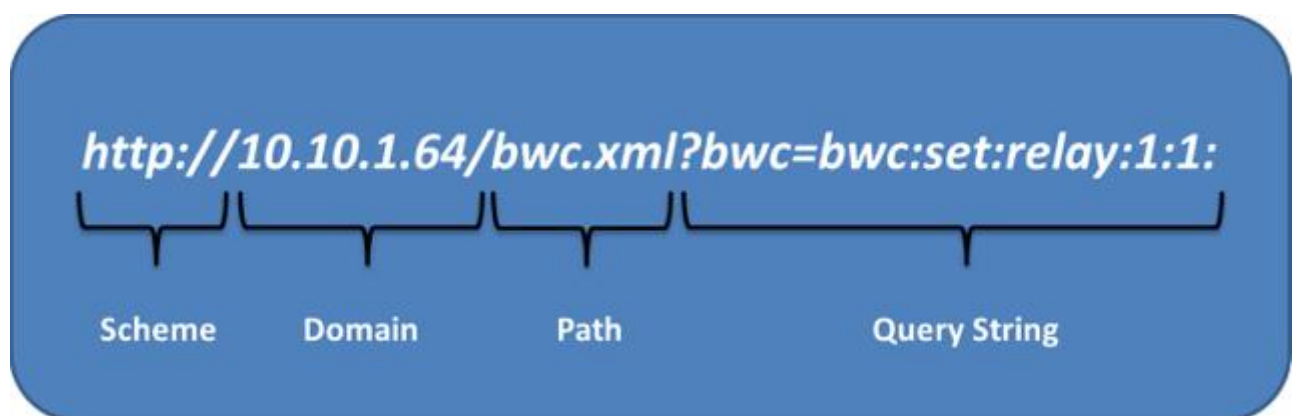
http://10.10.1.64/bwc.xml?bwc=bwc:set:relay:1:1:

To turn the relay off, send the string:

http://10.10.1.64/bwc.xml?bwc=bwc:set:relay:1:0:

Taking a look at the strings above, you may notice they are nearly identical. The only thing that changes is the last parameter which changes from a 1 to a 0. Now, let's break the string down so we can determine the best way to recreate it in our script device.

URL Components



Most of the time, you will be using the same root for all your http commands, we'll refer to this here at the URL Header.

URL Header

http:// - All http commands will begin with http:// (no surprise there) Note: https is not supported.

Domain (Host Name/IP Address) - The next component will typically be the IP address or the host name for the device you are communicating with. In some cases, you may need to pass a username and password in the URL String. This can be done by separating the username and password by a colon and following with the @ symbol.

Example: **http://username:password@192.168.1.25/path?query_string**

Port - A port number may be specified following the host name. This is optional and defaults to port 80 if not specified.

Example: **http://username:password@192.168.1.25:5425/path?query_string**

URL Path and Query String

Path - The path is used to specify the resource on the server. This is often the same for all commands and may be part of the header string if it does not change.

In the BC4 example above, the path is **/bwc.xml**

Query String - This is what we typically think of as the command. The query string contains the data to be passed to the server.

In the the BC4 example above, the query string is **?bwc=bwc:set:relay:1:0:**

Define URL Components in a Script Device

Begin by adding a Script Device to a project. To keep this simple, we will do this on a BC4 controller.

User Settings

Script Device: BC4_HTTP_Control

Device Name: BC4_HTTP_Control Protocol: API Port: IP:

Functions Incoming Data Feedback Shared Functions User Settings Notes

Name	Value
IP_Address	"10.10.1.64"

Settings Notes

Edit the vale for IP_Address to match the IP for the system. The IP address is a string and must be in quotation marks.

Settings Object Parse OK

Settings Object Result - (Read-Only, for Developer Use)

```
1 THIS_DEVICE.SETTINGS = {};  
2 THIS_DEVICE.SETTINGS.IP_Address = "10.10.1.64"
```

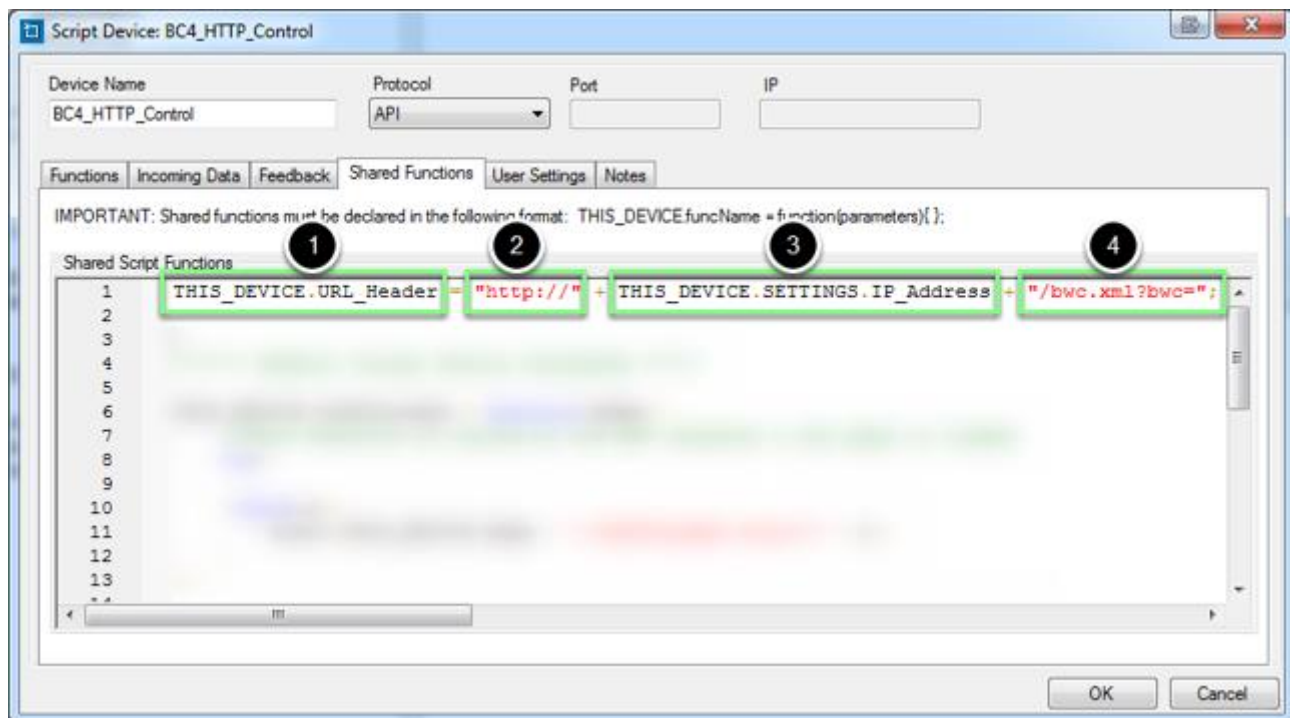
Settings must have Javascript-safe names. Values use Javascript variable declaration syntax. For example, string values must be wrapped in quotes. You may also use numeric, boolean, object, and array declarations.

OK Cancel

Make sure the object is valid after editing user settings.

When working with a device that has a host that may change (like an IP address for example) it's a good idea to create a user setting for this value. This way, less experienced users can easily modify the driver without needing to edit the actual script.

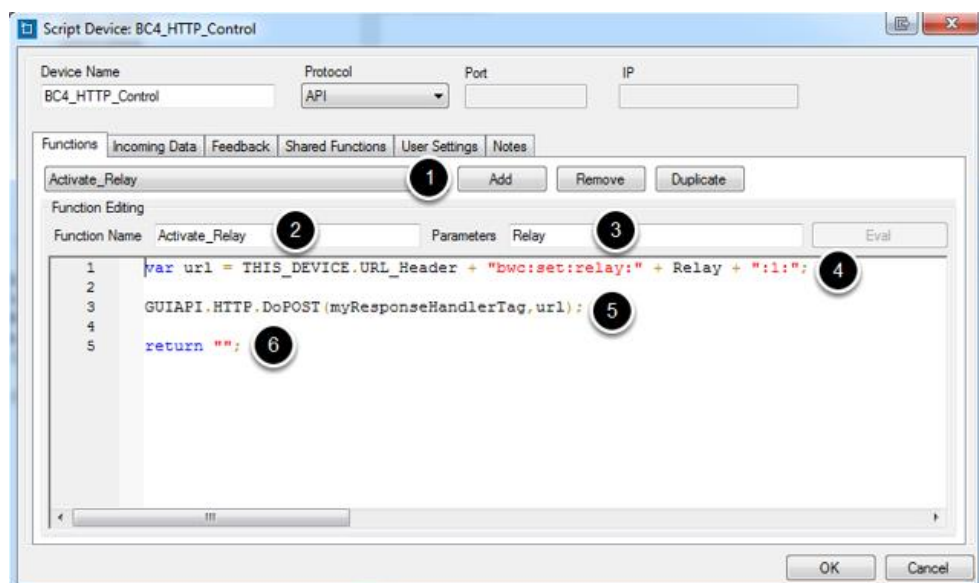
Shared Functions



The shared functions area is a great place to define data that will be global for the device and is not subject to change. This is where we will define our URL header and will extend it of the "THIS_DEVICE" object.

1. Define the header as an extension of the script device object "THIS_DEVICE"
2. Begin the string with ["http://"](http://)
3. Add IP address by referencing the IP_Address that was defined in the user settings tab. This could be hard coded as well if the host will not change.
4. Add the path to the string. In this case, we added part of the Query string (**?bwc=**) because it will not change throughout the protocol.

Functions



Now that we have the components, we are ready to start building functions (a.k.a. commands) by putting the chunks together.

1. Add a new Function
2. Assign a meaningful name
3. Declare parameters if needed
4. Build URL string and assign to a variable
5. Call the HTTP API function to send the command string (The API function also requires us to pass a callback function for the response. more on this later).
6. Return an empty string to return control to the higher function.

This step introduces several new concepts that have not been explained. Don't worry too much about the API command at this point, the main thing to take away is the syntax for sending an http string. This example uses the Post method. Some devices will require the Get method which is also available and would look like this:

`GUIAPI.HTTP.DoGET(myResponseHandlerTag,url);`

Code from function:

```
var url = THIS_DEVICE.URL_Header + "bwc:set:relay:" + Relay + ":1:";
GUIAPI.HTTP.DoPOST(myResponseHandlerTag,url);
return "";
```

Important Note: When using the API Protocol method of a Script Device, functions will not evaluate properly. Do not use Eval to test your functions!

Callback Function

When sending a command to a server via http, there will most likely be some sort of response. We must handle this response in some way, even we just discard it. In the string above, myResponseHandlerTag is a reference to a function we will define to handle the response. In the example above, we are simply printing the response to the debug window in Project Editor.

Here is the actual code you can use:

```
var myResponseHandlerTag = "MyResponseHandler";
GUIAPI.HTTP.AddHandler(myResponseHandlerTag,function(response) {
    try{
        GUIAPI.Debug.WriteLine(unescape(response));
    }catch(e) {
        alert(e);
    }
});
```

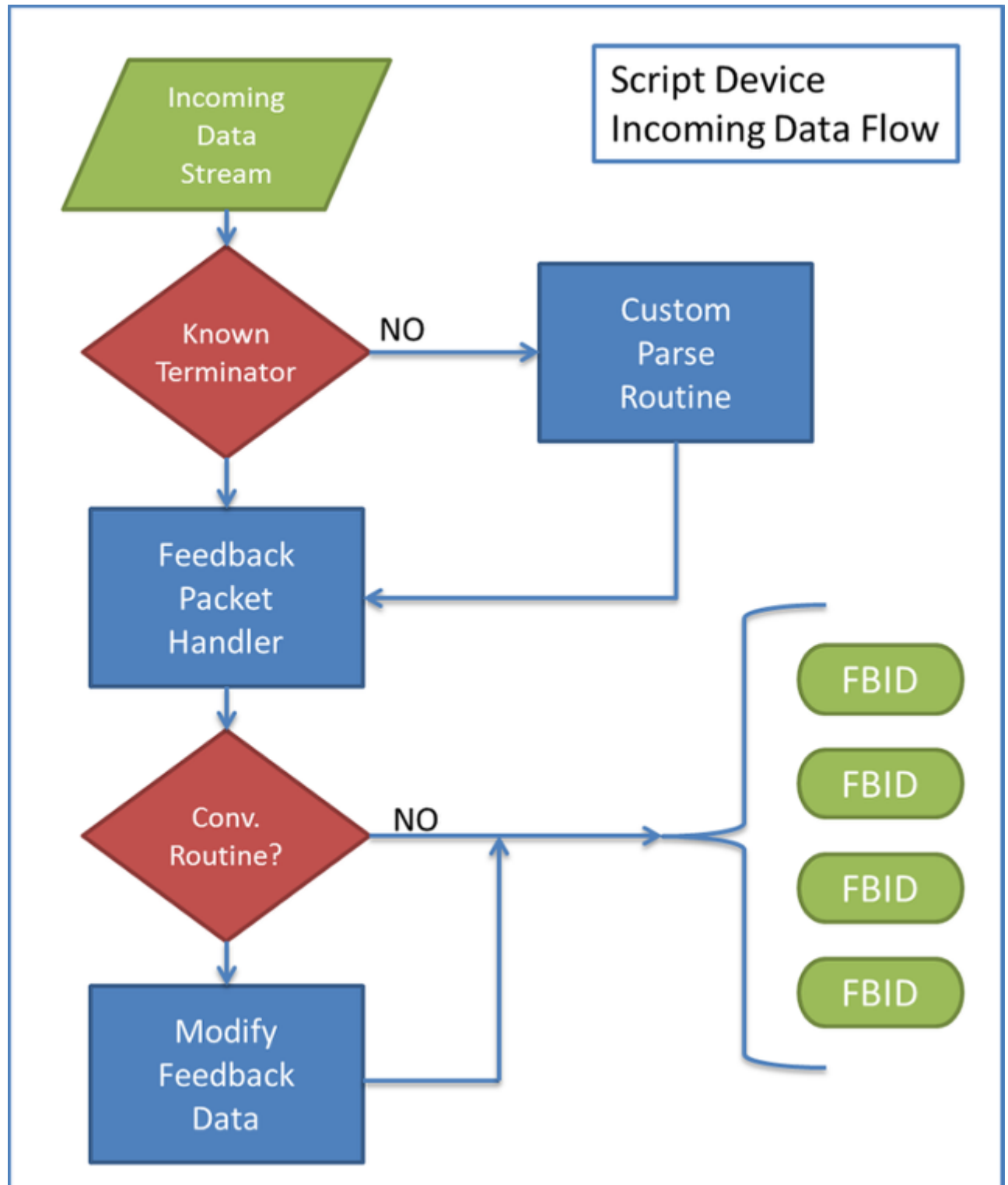
Next Steps

You have now seen how to send basic http commands from the app in a BC4 project. The technique is very similar with BC1 and BC2, however, because our advanced controllers do run JavaScript, additional considerations must be made to make sure the proper code runs on the local device. This information will be covered in the next installment.

Processing Feedback

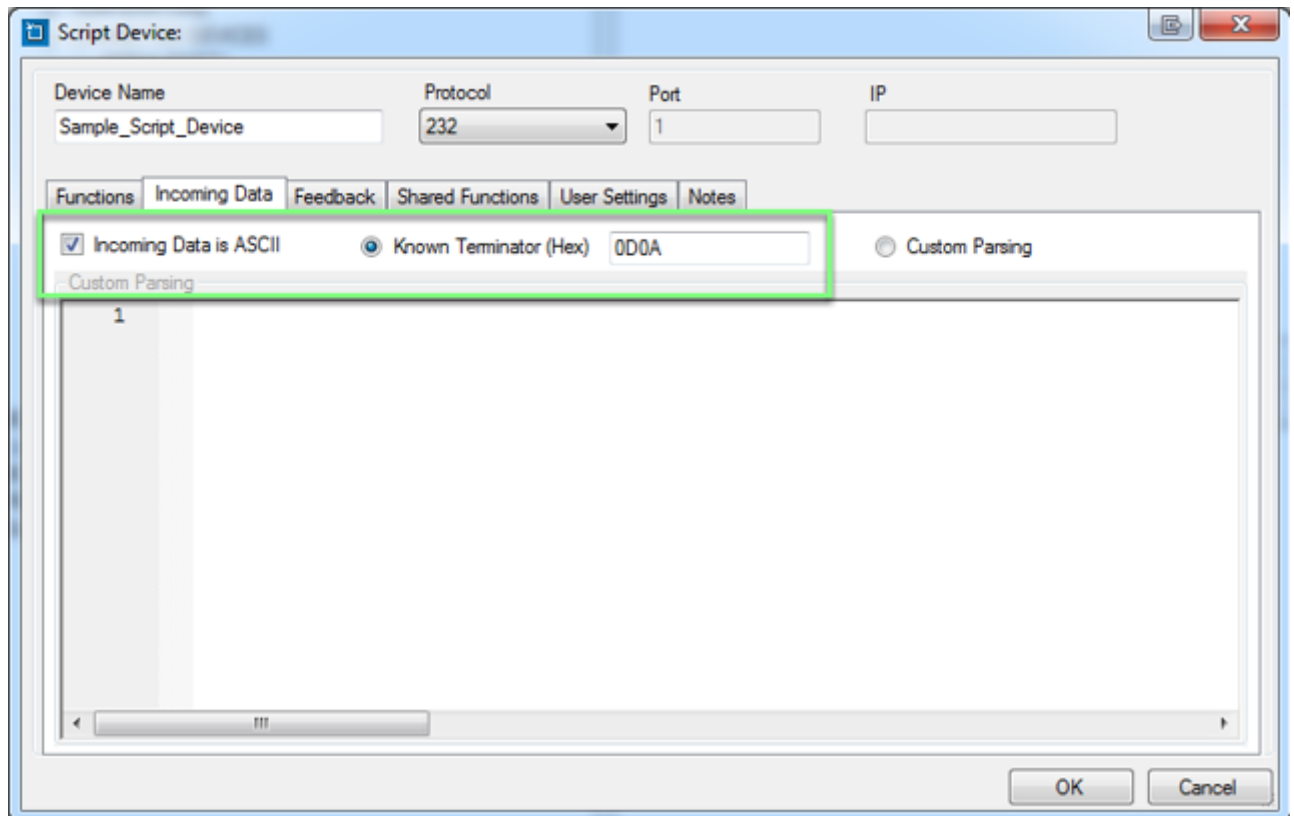
In the [Writing Commands](#) article you learned how to create simple strings using JavaScript. This article will help you understand how incoming data is processed and broken into meaningful information that can be displayed on a GUI interface as two-way feedback. The information provided here should be enough to get started creating some basic feedback but is not exhaustive by any means. Advanced developers will want to refer to the JavaScript API for additional details.

Data Flow



The diagram above show how incoming data from RS-232, a TCP Socket or a UDP datagram is processed (http responses are handled differently).

Incoming Data



The first step to parsing incoming data is to know when you have a complete packet that should be processed. In most cases, incoming data will have a known terminator like a Carriage Return, a Line Feed or both. In cases where there isn't a known terminator, it will be necessary to create a custom parse routine to determine when a packet is complete and ready to be processed. This will vary greatly depending on the protocol of the device you are working with, some common methods are:

- Known length based on a standard length or a length byte in the start of the packet
- Period of inactivity - Process data after a predefined gap in the incoming data flow
- Any other method where a reliable pattern can be determined.

For the remainder of this article, we will only consider data with a known terminator. These options are set on the **Incoming Data** tab of the Script Device (see image above).

Incoming Data is ASCII - This box is checked by default and all incoming data is handled in ASCII format. If you are working with a protocol that uses unprintable characters or extended ASCII, you may wish to un-check this option in order to treat incoming data as HEX rather than ASCII. In this mode, the data are represented in ASCII HEX format. For example, 'ABC' [3 bytes] becomes '414243'[6 bytes].

Known Terminator - When selected, the byte value(s) listed in the field will be used to determine when a packet should be sent to the feedback handler. Carriage Return + Line Feed (0D0A Hex) are the default setting and are very commonly used.

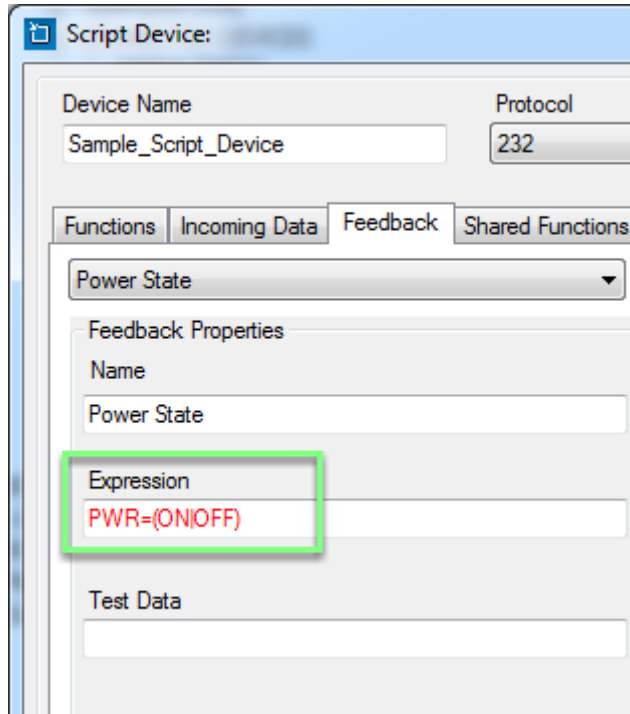
Feedback

Now that the incoming data stream is being broken into packets, it's time to filter these packets for meaningful data.

Feedback is comprised of 3 components.

1. Expressions
2. Conversion Routines
3. Matches

Expressions



Project Editor uses [Regular Expressions](#) to compare packets against patterns of characters and capture matches.

Note: The expressions are created by the constructor function so there is no need to use the literal format of: **/expression/**

Example

The expression should contain the text you expect to see for the data we you interested in.

In this example, the AV Receiver returns two different strings for the power state **PWR=ON** or **PWR=OFF**

Based on this information, we need to filter any string containing **PWR=**

In the image above, this is how the expression begins. To extract the part of the string we are interested in, we use parenthesis to create a "Capture Group". In this case, the capture group contains two literal strings separated by the pipe symbol **/**. The pipe represents an OR so the expression above will capture ON or OFF but nothing else. There are several different ways to capture this same data but it's a good practice to make your expressions as strict as possible to keep from capturing unexpected results. Here are a few other ways we could capture this same data from least restrictive to most restrictive.

1. **PWR=(.*)**
2. **PWR=([A-Za-z]+)**
3. **PWR=([A-Z]+)**
4. **PWR=(ON|OFF)**
5. **^PWR=(ON|OFF)**

Character Class [XYZ]

In example expression 2, the square brackets inside the parenthesis represent what is known as a character class. This is very useful when the incoming data has more than just a few options and you don't want to list all the literal possibilities. Character classes define a range of characters that are allowed in the capture group.

Sample character classes:

- ([A-Z]) Any single upper case letter from A-Z
- ([A-Za-z]) Any single upper or lower case letter from A-Z
- ([0-9]) Any Single Digit
- ([0-9]{2}) Exactly 2 digits
- ([A-F0-9]{2}) Exactly two characters 0-9 or A-F (Think Hex byte)
- ([A-Za-z0-9]{1,12}) Match A-Z or a-z or any digit 1 to 12 times

Using Special Characters

The examples above work fine but can be simplified further using special character patters shown below.

- \d = digit [0-9]
- \D = non-digit [^0-9]
- \w = word character [0-9a-z_A-Z]
- \W = non-word character [^0-9a-z_A-Z]
- \s = a whitespace character [\t\n\r\f]
- \S = a non-whitespace character [^ \t\n\r\f]

Rewriting the examples above we have:

- (\w) Any single upper case letter from A-Z
- (\w) Any single upper or lower case letter from A-Z
- (\d) Any Single Digit
- (\d{2}) Exactly 2 digits
- ([A-F\d]{2}) Exactly two characters 0-9 or A-F (Think Hex byte)
- (\w{1,12}) Match A-Z or a-z or any digit 1 to 12 times

Note: We don't even need the square brackets if there is only one data type.

In example 5 above, notice the expression starts with a caret "^". This tells the expression to only match from the beginning of the string.

PWR=ON (Is a match)

MZ PWR=ON (NOT a match)

Depending on the situation this may not be desirable but it does increase overall performance when used.

We will explain how to test your expressions once we set up the Matches.

Here are a few useful resources for more learning about Regular Expression syntax.

- http://www.eclipse.org/tppt/home/downloads/installguide/qla_42/ref/rregexp.html
- http://www.w3schools.com/jsref/jsref_obj_regexp.asp
- <http://rubular.com/>

Matches

Functions Incoming Data Feedback Shared Functions User Settings Notes

Zone 1 Feedback Add Remove Duplicate

Feedback Properties

Name

Zone 1 Feedback

Expression

#Z1,(ON|OFF),SRC(d),VOL(d+),DND(0|1),LOCK(0|1)

Test Data

#Z1,ON,SRC4,VOL60,DND0,LOCK0

Matches Add Remove

	Name	FBID	Test	ConvertedTest
▶	Zone 1 Power	NE6G_Z1PWR	ON	
	Zone 1 Source	NE6G_Z1SRC	4	Source 4
	Zone 1 Volume	NE6G_Z1VOL	60	-60
	Zone 1 DND	NE6G_Z1DND	0	
	Zone 1 Lock	NE6G_Z1LOCK	0	

Conversion Routine (return converted 'data' variable as string)

1

OK Cancel

Each capture group in an expression needs a match. Most of the time there only be one match from a particular packet but there are cases where multiple items can be captured from a single packet.

Click the Add button to the right of the matched label to create a new match. The first match will correspond to the first capture group in the packet from left to right.

Name - Assign each match a name. This name will be displayed in the feedback assignment drop down menus.

FBID - Assign each match a unique Feedback ID. FBIDs are basically global variables and must be unique throughout the entire project file. Try to follow a convention that is not likely to be duplicated. FBIDs must also follow the rules for JavaScript names. Use only characters A-Z (upper or lower), the underscore character and digits, however, do not begin names with a digit.

Test - This column will display the captured result of the test data after it has gone through the expression. This is a useful tool for testing your expression syntax.

Converted Test - Sometimes the captured data will need to be modified before assigning the value to a feedback item.

Conversion Routine

Matches

Add

Remove

	Name	FBID	Test	ConvertedTest
	Zone 1 Power	NE6G_Z1PWR	ON	
▶	Zone 1 Source	NE6G_Z1SRC	4	Source 4
	Zone 1 Volume	NE6G_Z1VOL	60	-60
	Zone 1 DND	NE6G_Z1DND	0	
	Zone 1 Lock	NE6G_Z1LOCK	0	

Conversion Routine (return converted 'data' variable as string)

```
1 // Edit Source Names in statement below
2 // to match installed system
3 switch (data){
4     case "1":
5         return "Source 1";
6     case "2":
7         return "Source 2";
8     case "3":
9         return "Source 3";
10    case "4":
11        return "Source 4";
12    case "5":
13        return "Source 5";
14    case "6":
15        return "Source 6";
16    default :
17        return "Source Unknown";
18 }
19 return "";
```

The conversion routine is an optional element of a feedback packet.

Every match can have it's own conversion routine.

Any valid JavaScript can be used in the conversion routine including accessing shared functions and API methods.

A simple example would be adding units to a volume response or converting an ON/OFF text response to a binary digit response to use on a button state. It's even possible to trigger a macro based on feedback if the Script Device is running on a BC1 or BC2 Controller. (Remember...BC4s don't run JavaScript!)

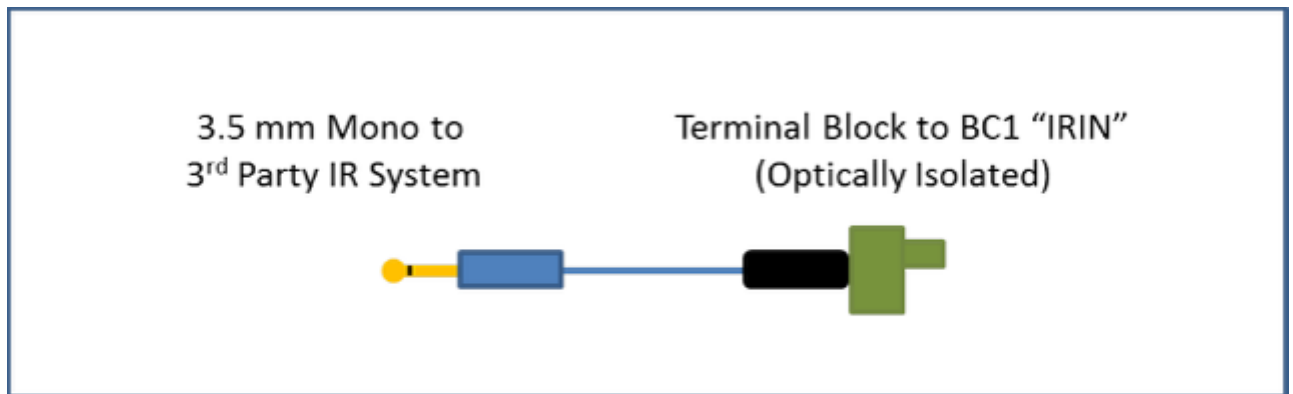
BC1 IR Input

All BC1s have a 3 pin header labeled **IRIN**. This input can be used to send IR commands to a BC1 in order to trigger macros from a third party control system. This article will guide you through connecting and programming a BC1 for this purpose.

Requirements

- Generation 1 or 2 BC1 Controller
- BC1 IR Adapter
- External IR System Capable of Sending learned or imported IR Codes
- Project Editor 1.9 or later Recommended

BC1 IR Adapter



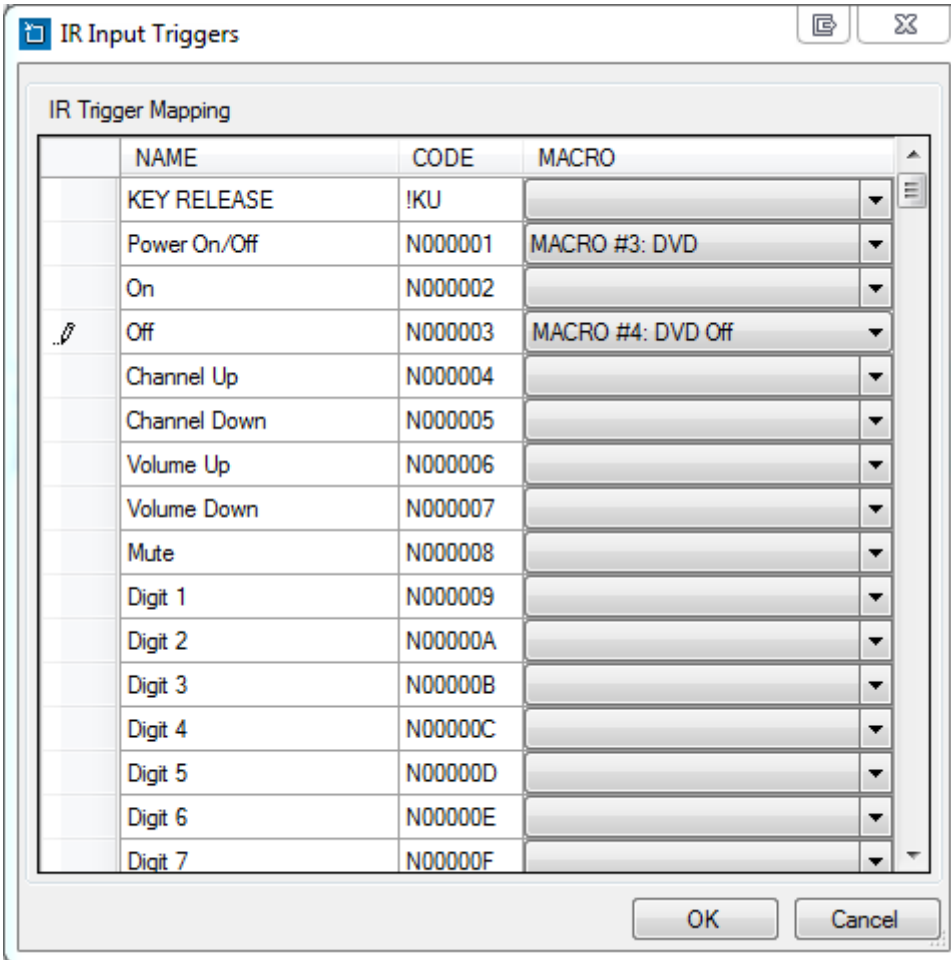
BC1 IR Adapter

This adapter is used to connect the BC1 to an external 3rd party IR system. This might be an RF base station or an IR distribution system with IR receivers in the room where the IR remote is used. This adapter is optically isolated to eliminate any ground potential that may exist between the external system and the BC1.


Trigger Codes

Now that we have connected the BC1 to the third party IR system we need to get the IR codes to the third party system. The BC1 only responds to IR trigger codes from NEC Codeset Zero. [This codeset can be downloaded in a variety of formats here.](#) Once you have downloaded the linked zip folder, extract the files inside and use the appropriate codes for your third party system. Codes are provided in CCF, RTI and CSV formats. There is also a text file explaining all 255 codes and how their names map to Project Editor.

Program IR Input Triggers



The screenshot shows a window titled "IR Input Triggers" with a sub-header "IR Trigger Mapping". It contains a table with three columns: NAME, CODE, and MACRO. The table lists various IR triggers and their corresponding codes and macro assignments. The "MACRO" column has dropdown menus for each row. The "Off" row is currently selected, showing "MACRO #4: DVD Off".

	NAME	CODE	MACRO
	KEY RELEASE	!KU	
	Power On/Off	N000001	MACRO #3: DVD
	On	N000002	
	Off	N000003	MACRO #4: DVD Off
	Channel Up	N000004	
	Channel Down	N000005	
	Volume Up	N000006	
	Volume Down	N000007	
	Mute	N000008	
	Digit 1	N000009	
	Digit 2	N00000A	
	Digit 3	N00000B	
	Digit 4	N00000C	
	Digit 5	N00000D	
	Digit 6	N00000E	
	Digit 7	N00000F	

At the bottom of the window are "OK" and "Cancel" buttons.

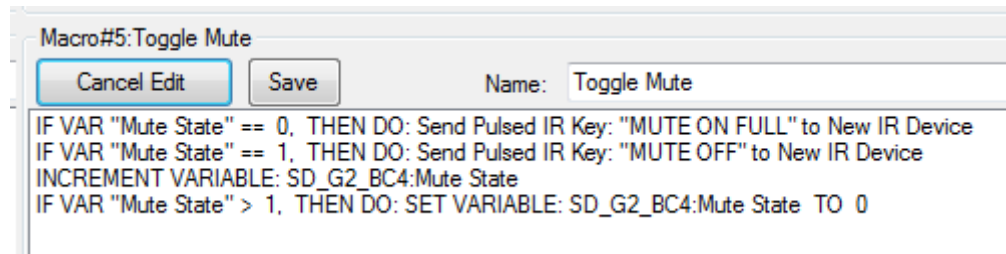
Before completing this next step make sure you have already programmed and tested all your BC1 macros.

Under the BC1MACROS node of the project tree you will see an IR TRIGGERS Node. Right click this node to open the properties for the IR Input Triggers. All you have to do now is choose the corresponding macro for each programmed IR Trigger Code. Once you have selected all the appropriate macros, click done and upload the BC1 to store the changes. Send IR codes from the 3rd party system to test the macros activate as expected.

How do I create a toggle from two discrete commands?

It seems most of the time we are looking for discrete control but there are still situations when it's useful to have a button toggle between two commands with each button press. This short article will show you how to use a macro and a user variable to toggle between two discrete commands with each actuation of the macro.

Example



This example uses a BC4 but the concept applies to BC1s and BC2 as well.

Scenario: You have a device that only has Mute On and Mute Off commands but you would like to use one button to toggle between states.

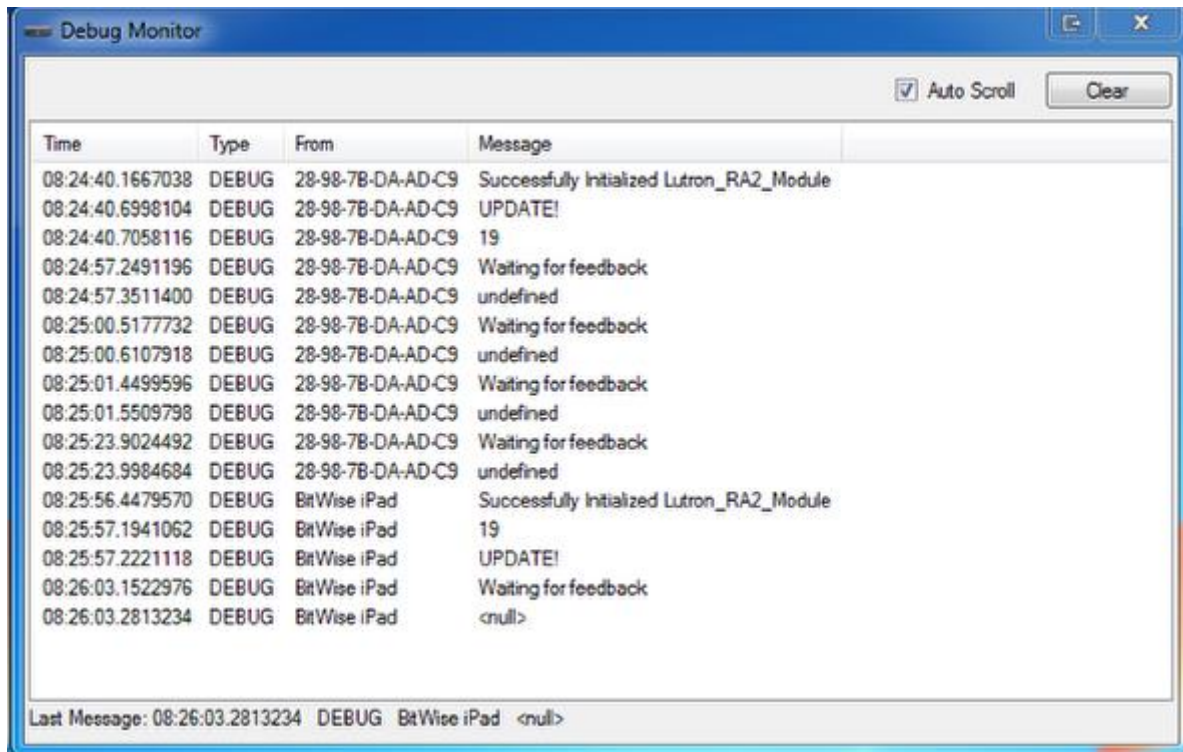
Solution: Use a macro and a user variable to track which command was sent last and evaluate to send the opposite command.

1. Define a user variable - In this case I am going to name the variable "Mute State"
2. Add a New Macro - I called mine "Toggle Mute"
3. Add the two discrete commands to the macro -The non-normal state should be first (i.e. Mute On in this example)
4. Add an evaluation to the first command - Double click the first command (Mute On) and check against the user variable. We have done nothing to set the variable so it should be equal to zero and if it is let's send the Mute On command.
5. Add an evaluation to the second command - Since we are running the Mute on if the variable is equal to zero then we should turn mute off the user variable equals one.
6. Increment the user variable - The first time the macro runs the user variable will be zero, we will increment the variable by one so that the correct command will run the next time the macro is called.
7. Set the user variable back to zero when greater than one. Since we are changing the user variable with the increment command we need to prevent it from getting larger than one. These last two steps essentially force the user variable to toggle between zero and one each time the macro is executed.

If you have followed these steps, you should have a macro that looks something like the image above.

Important note: User variables are set back to zero after an upload or after a reboot for BC4's.

Debug Monitor



Project Editor 1.8 introduced a new tool called the "Debug Monitor". Although primarily intended to aid with driver development, this new tool can also be very useful for troubleshooting projects. The Debug Monitor will capture and display many standard events as well as any data sent to the new built in Module API functions for debugging. Since these broadcasts are non-blocking, they don't interrupt the flow of a program the way a JavaScript alert() would.

The sample screen shot above shows a device file successfully loading but later having errors due to undefined variables when pressing buttons.

Messages can be printed to the debug monitor from the app or a controller (BC1 or BC2) using JavaScript. This can be a useful way of determining the value of a variable or a feedback packet.

Example code that could be placed in a Script Device.

```
if(ON_CONTROLLER){  
  
    CTRLAPI.Debug.WriteLine("Debug Message goes Here");  
  
}  
  
if(ON_GUI){  
  
    GUIAPI.Debug.WriteLine("Debug Message goes Here");  
  
}
```

To access the debug monitor, go to Tools -> Debug Monitor.

Factory Default Procedure - BC1 and BC2

The BC-1 and BC-2 have two levels of factory default. For the remainder of the document, we will refer to both as the BC-1. The first default level (Soft Reset) is similar to a BC-4 default and will only reset the networking information and return the unit to DHCP mode. The second level (Hard Reset) will completely erase any existing project related files and will restore its' firmware to factory default from an internal archived copy.

For either procedure, you will need a paper clip or a pen to reach the recessed reset button next to the 12V connector. If you have trouble getting the reset button pressed quickly enough, you can hold it down before applying power to the BC-1.

Soft reset - Quickly return to DHCP and clear network settings

1. Remove power from the BC-1 by unplugging the +12 VDC connector
2. Replace the power connector and press/hold the reset button for 10-15 seconds. Note: Reset button must be pressed within a few seconds of applying power.
3. Watch for the Red Status light on the front to illuminate and extinguish within 15 seconds of power up. Once this has occurred, you can release the reset button and allow the BC-1 to finish starting.
4. Your BC-1 should now be returned to DHCP mode.

Note: Make sure not to hold the button for longer than 15 seconds or you might accidentally perform a full factory default!

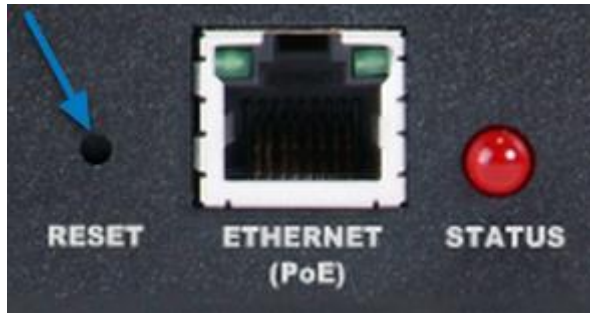
Hard Reset - This is not typically needed and should only be done on the advice of BitWise Technical Support

1. Remove power from the BC-1 by unplugging the +12 VDC connector
2. Replace the power connector and press/hold the reset button for 30-40 seconds. Note: Reset button must be pressed within a few seconds of applying power
3. Watch for the Red Status light on the front to begin flashing rapidly and then release the reset button. (Approx 30-40 seconds)
4. Wait while the BC-1 goes through the process of restoring itself from an archive file. The process is complete once you observe a solid red status light
5. The firmware will now be restored to the original version loaded during production. Check for firmware updates and load the newest firmware onto the BC-1
6. Power cycle the BC-1 to clear the memory and load the new files
7. You should now have a clean and updated BC-1

Factory Default Procedure - BC4

This article will explain the steps necessary to factory default a BC-4 Controller. This procedure will not erase macros or programming but will clear all networking information and return the unit to DHCP mode. This is especially useful when moving a BC-4 from one network to another or when having trouble "Discovering" a unit on the network.

You will need a paper clip or pen to reach the recessed reset button.



You will need a paper clip or pen to reach the recessed reset button.

Hold the reset button (left of the Ethernet port) for approximately 10 seconds. During this period, the Red Status light will:

- Go out
- Come back on
- Go out a second time

You can now release the reset button and allow the BC4 to finish booting normally.

Your unit should now be returned to DHCP mode and obtain an IP address from the Network Router.

BC4 Master Reset Procedure

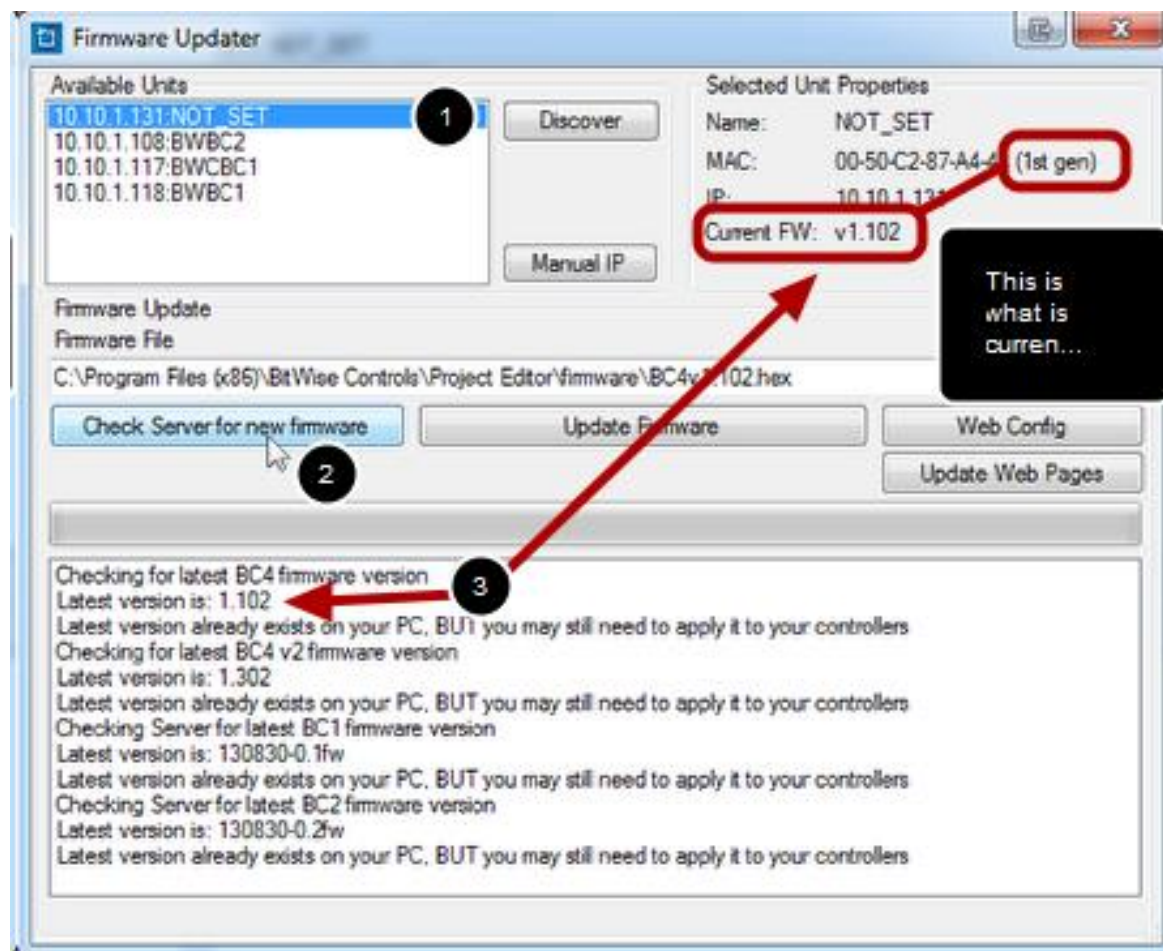
Purpose

This tech note will guide you through performing a full factory reset on a Gen 1 or Gen 2 BC4 Controller.

This procedure will likely fix the following issues.

- IR Version Unknown
- No IR Output from emitters
- Installed BCX Module shows "Unknown"
- No relay Control
- No Response from GPIOs
- Missing Web Page

Install Most Current Firmware Version



Current versions of software and firmware can be determined by going to **Help -> About** in Project Editor or by clicking the "Check Server for new firmware" button in the firmware updater. (If your Project Editor version is not current, you will not see the most current version in the About window.)

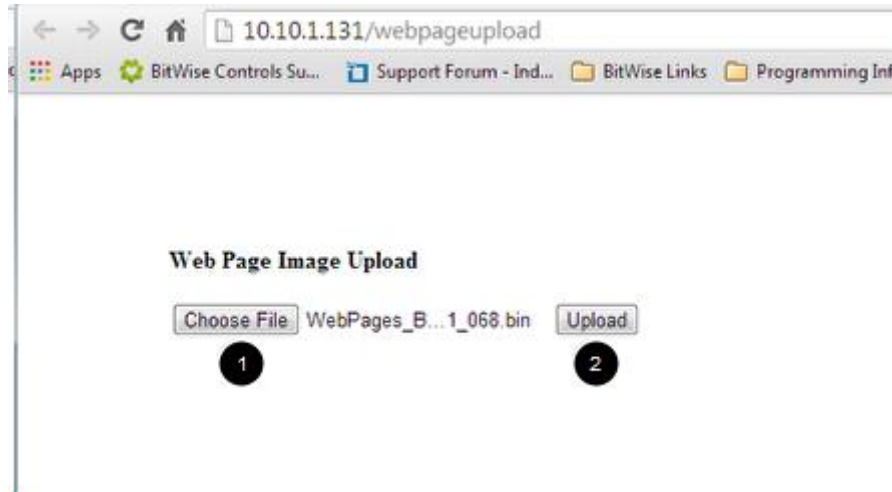
Open the Firmware Updater **Tools -> Firmware Updater** or Ctrl+w.

1. Select the IP address of the BC4 you are working with
2. Click the **Check Server for new firmware** button
3. Compare the Latest version for this BC4 to the Current BC4 FW (Generation Matters! Make sure to compare the generation too).

If you are on an older firmware version or want to reflash the current firmware (may not be a bad idea), Click the Update Firmware button and follow the onscreen prompts.
Don't forget to power cycle the BC4 after the firmware update.

Don't close the firmware updater yet, we are not finished. Update Web Pages

Update Web Pages



Depending on which firmware was previously on your BC4, you may need to re-load the BC4s internal web pages. There is no harm in re-loading the pages, it is recommended to perform this step to be certain the files are present.

In the same Firmware Updater window from step one, make sure the BC4 is still selected in **Available Units** and click the **Update Web Pages** Button. You will see a notification indicating the folder location of the web pages file. When you click **OK** to dismiss this notice, your default web browser will open to the IP address of the selected BC4.

1) Click the **Choose File** button and navigate to the file WebPages_BC4_1_068.bin file.

For 32 bit Windows Versions

C:\Program Files\BitWise Controls\Project Editor\firmware\webpages

For 64 Bit Windows Versions

C:\Program Files (x86)\BitWise Controls\Project Editor\firmware\webpages

2) Click the Upload Button

You should now see a page with the Words "Web Page Update Successful" and a link "[Site main page](#)". Click the link to make sure the pages loaded correctly.

Don't close the web page, we will use it in the next step.

Send Factory Config Command



Now that we have current firmware and web pages, let's force the BC4 to recognize the the X1 module that contains the IR, Relays, RS-232 and GPIOs.

Paste the following string after the IP address of the BC4 and press the Enter key.

/bwc.xml?bwc=bwc:set:factoryconfig:1111:

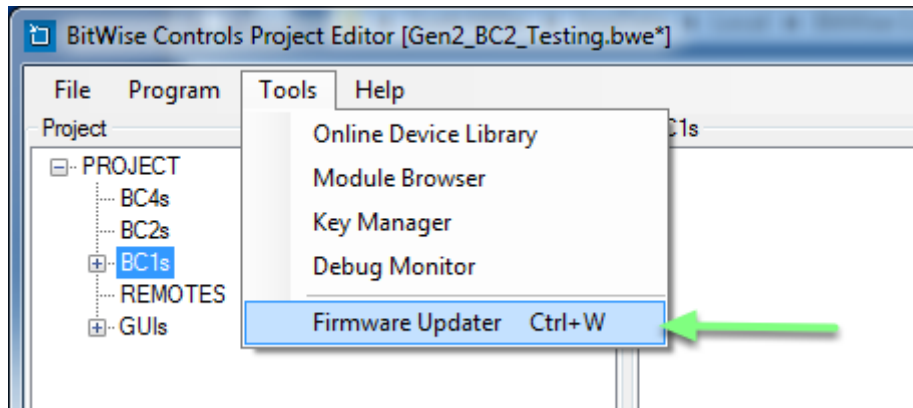
Your complete string should look like the image above and you should receive a similar response depending on the browser you are using.

Power cycle the BC4 again and it should be returned to full operation. If this procedure does not rectify your issue, please [open a support ticket](#).

Update Internal Web Pages

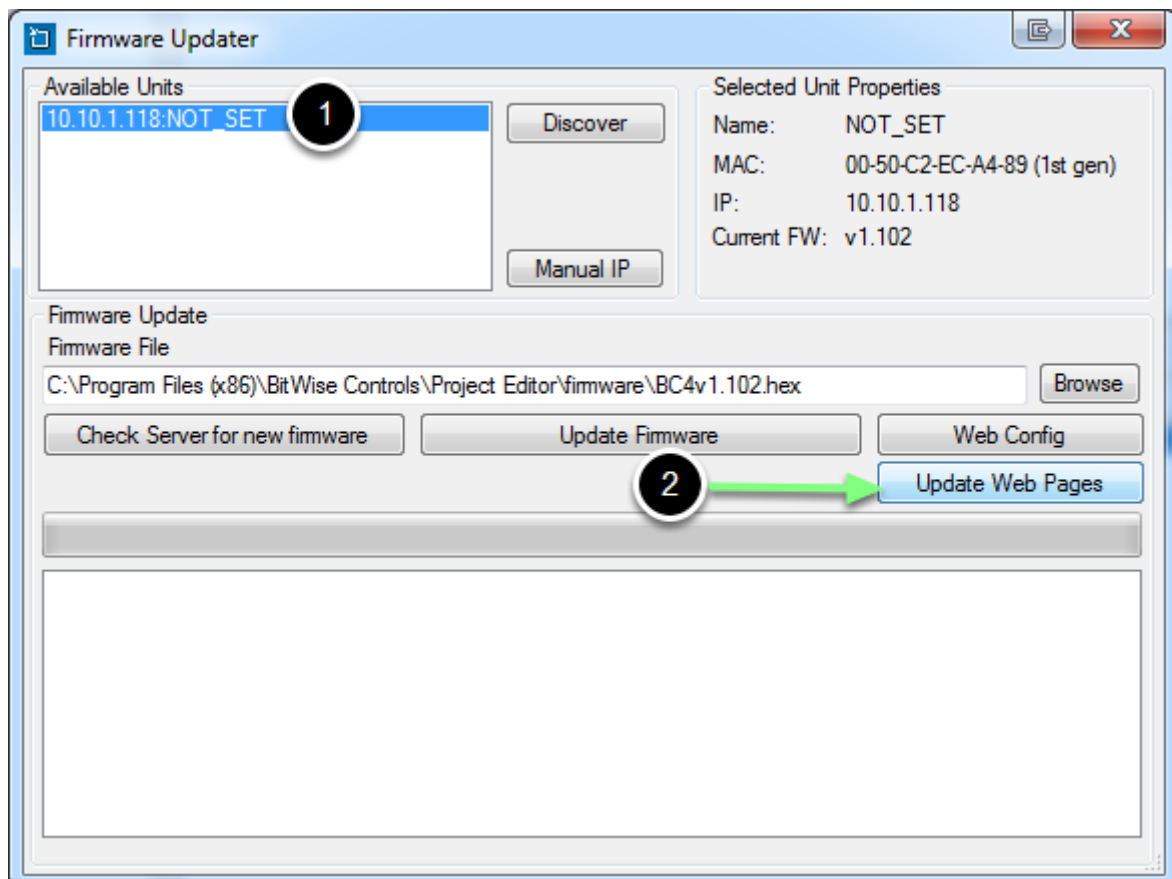
There may be cases where you will need to update the internal Web Pages for a BC4 Controller. The most common reason for this would be updating an older BC4's firmware from a version lower than 1.073.

Open the Firmware Updater



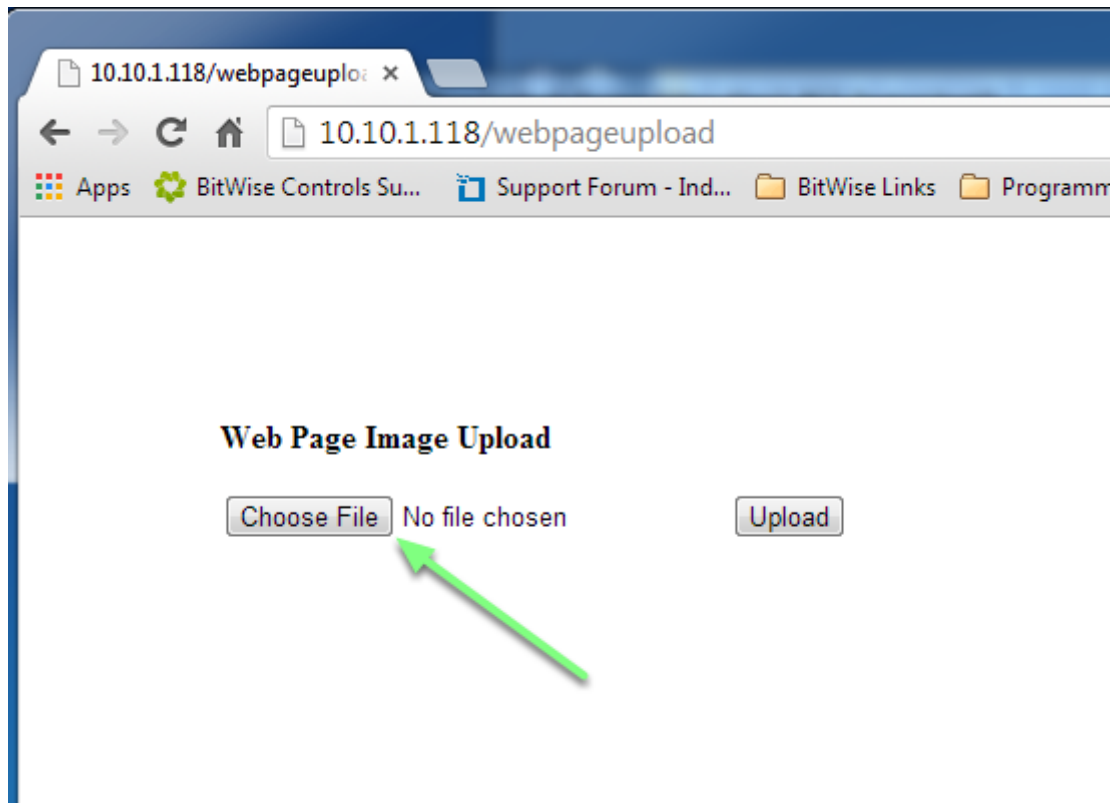
Go to Tools->Firmware Updater or press Ctrl+W on your keyboard.

Select the BC4 from the Available Units



1. Select the BC4 from the list of available units
2. Click the "Update Web Pages" button.
3. Note the alert that pops-up and click "OK" to advance. Your default browser will open the BC4's Web Page Upload tool.

Set the Web Pages File Path



Locate the file "WebPages_BC4_1_068.bin" using the file browser window that pops-up when the button is pressed then click "OK"

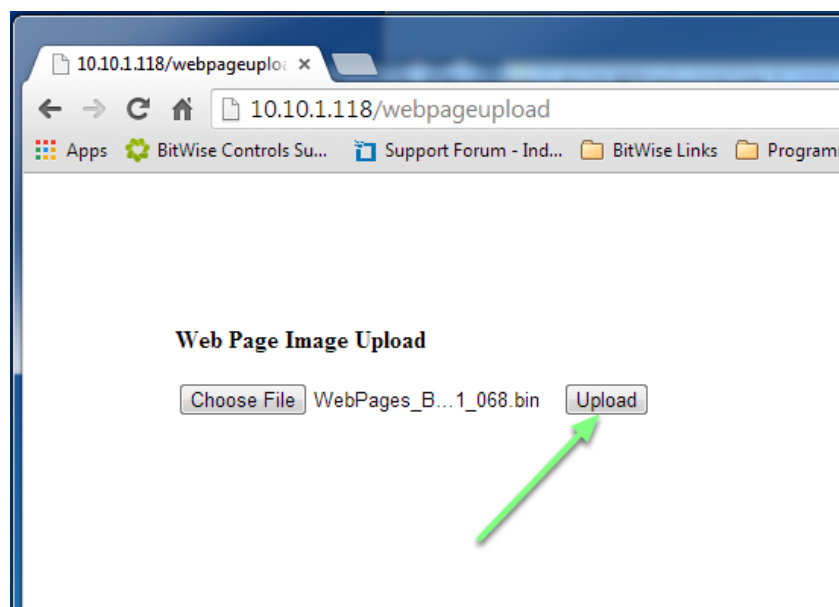
For 64 bit Windows versions, the file will most likely be located in:

C:\Program Files (x86)\BitWise Controls\Project Editor\firmware\webpages

For 32 bit Windows versions, the file will most likely be located in:

C:\Program Files\BitWise Controls\Project Editor\firmware\webpage

Upload the Selected File

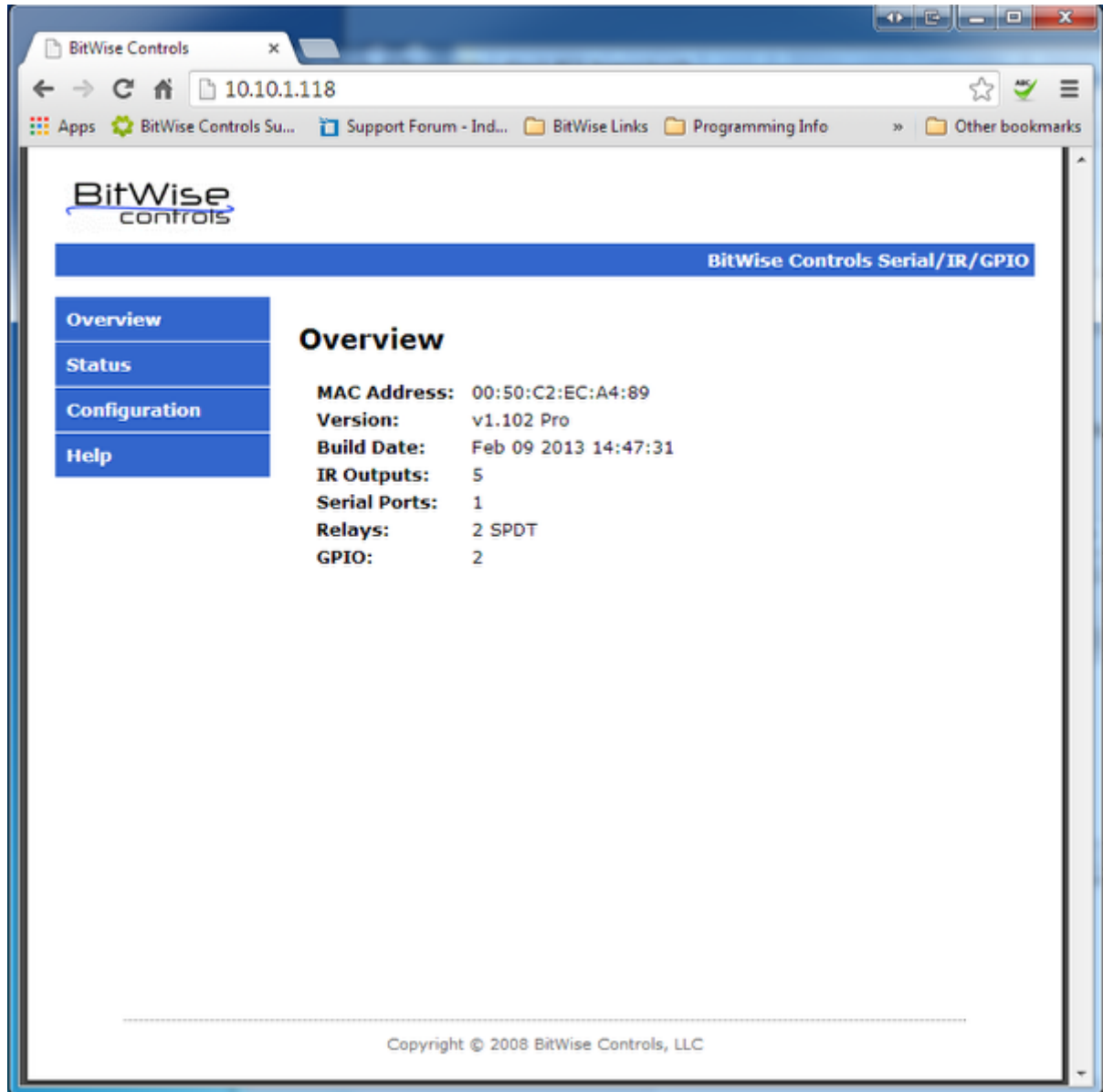


Click the "Upload" button to load the file to the BC4. Wait for the page to reload with:

Web Page Update Successful

Site Main Page

Verify Web Page



Click the "Site Main Page" link and verify the web pages loaded correctly.

You have now updated the BC4's internal web pages. Power cycle the BC4 to refresh all internal settings.

Firmware Recovery - BC4

If a BC4 firmware update is interrupted in progress, the unit may become unresponsive and require the firmware to be restored. The following conditions indicate that a BC4X1 unit requires a firmware recovery.

- The red “Status” LED is off, and the left green network activity LED is blinking.
- The unit cannot be “pinged”, or discovered by BitWise software.

When in this state, the unit will return to its default IP address of 192.168.10.10, and await a firmware upload. To recover the unit, use the following procedure

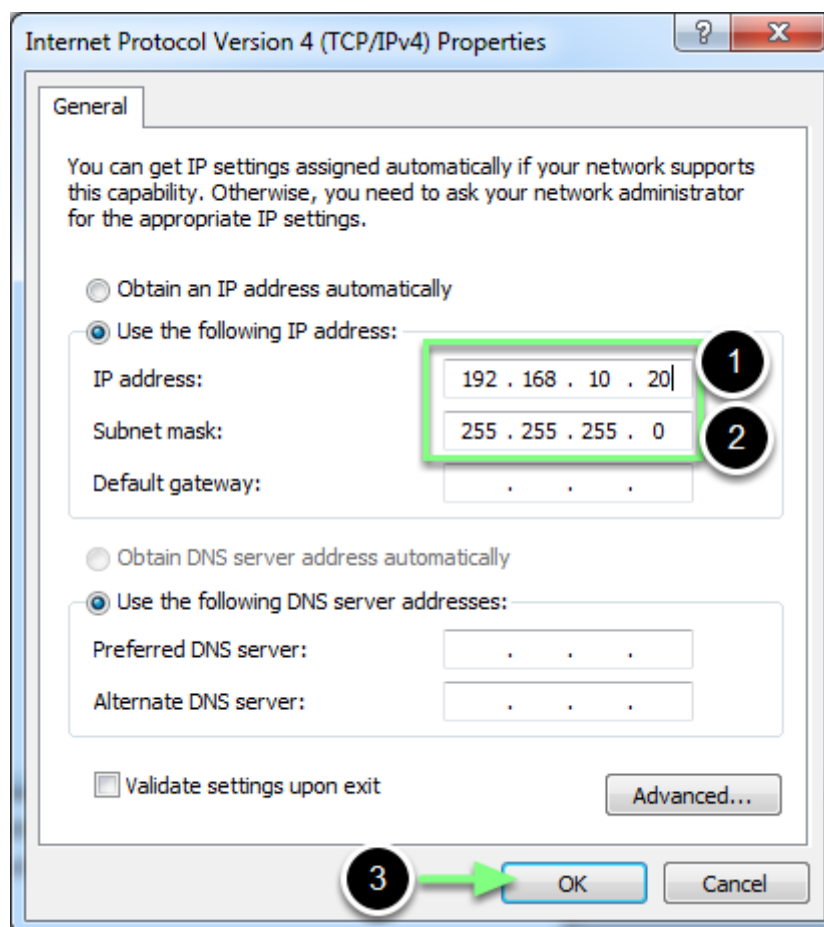
Connect the BC4 directly to a Computer

Connect the BC4 directly to your PC (one that has Project Editor installed of course) with a known good Ethernet cord. You are going to bypass all potential hubs, switches, and wireless issues by directly connecting to the BC4 unit. You should have link lights on both the BC4 and your PC.

If you do not have link lights:

- Option 1: You may need to use a Crossover Cable http://en.wikipedia.org/wiki/Ethernet_crossover_cable
- Option 2: Connect both the BC4X1 unit and your PC into a standalone hub or switch that is not connected to anything else

Change your PC's Network Settings



Before making changes to your network settings, you may want to take a screen shot if you have any non-standard settings or a static IP already set.

Temporarily assign your PC to a static IP of 192.168.10.20 and a subnet mask of 255.255.255.0

<http://windows.microsoft.com/en-HK/windows7/Change-TCP-IP-settings>

You will not need gateway or DNS settings while in this temporary state. Reminder: You will not be able to ping the BC4X1 unit at this point.

Disable Firewalls and Anti-Virus Software (optional)

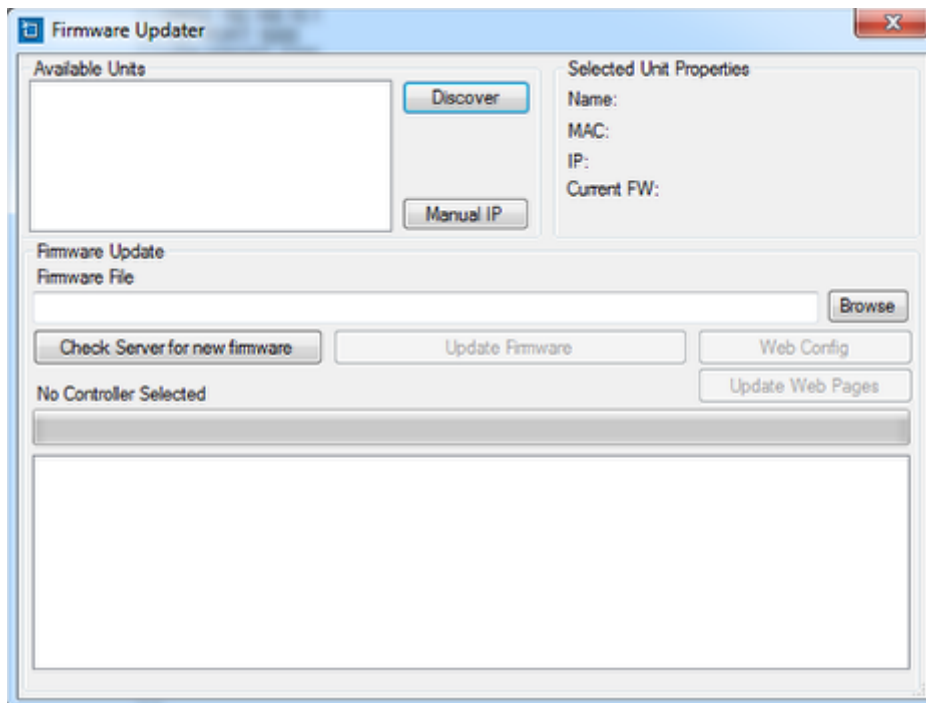
Temporarily disable all Firewalls and Antivirus Software on your PC

Optional Step: Boot your PC in “Safe Mode with Networking”

<http://windows.microsoft.com/en-HK/windows7/Advanced-startup-options-including-safe-mode>

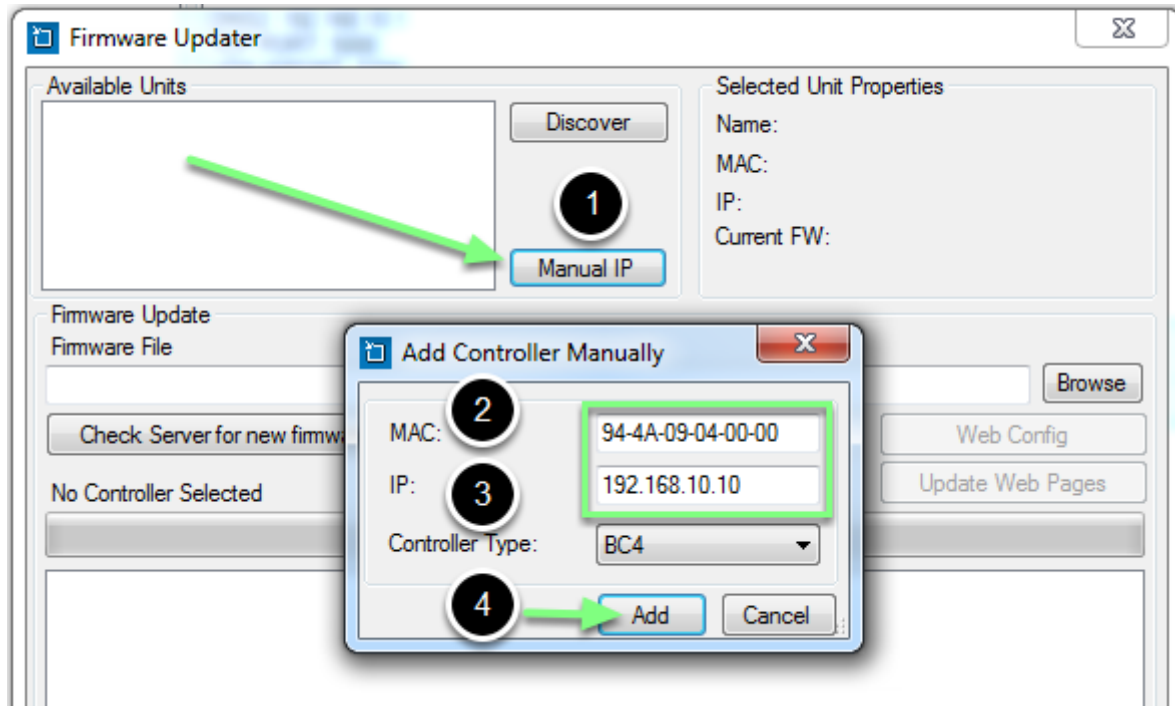
Hint: Press the “F8” Key before Windows starts

Open Firmware Updater



Open “BitWise Project Editor” and select “Firmware Updater” from the “Tools” menu. (Ctrl+W)

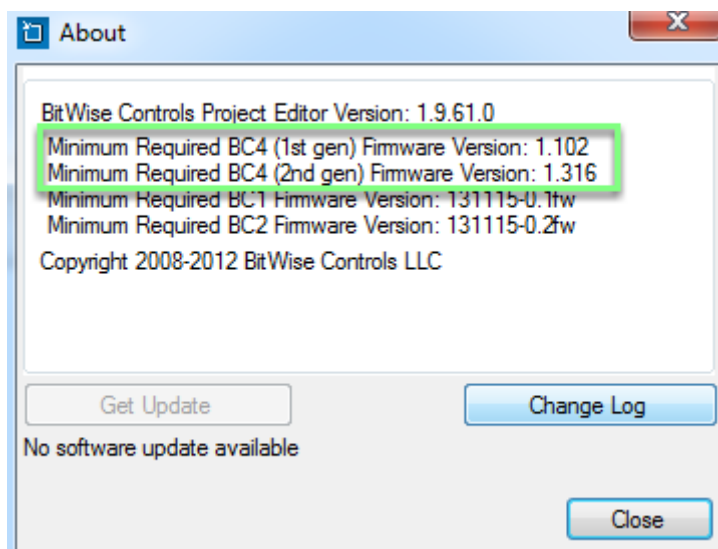
Add the BC4 Manually



1. Use the "Manual IP" button to manually add the BC4 unit to the list
2. Enter the MAC address of the unit you are recovering
3. Leave the default address 192.168.10.10
4. Click "Add"

Note: "Discover" will not work at this point.

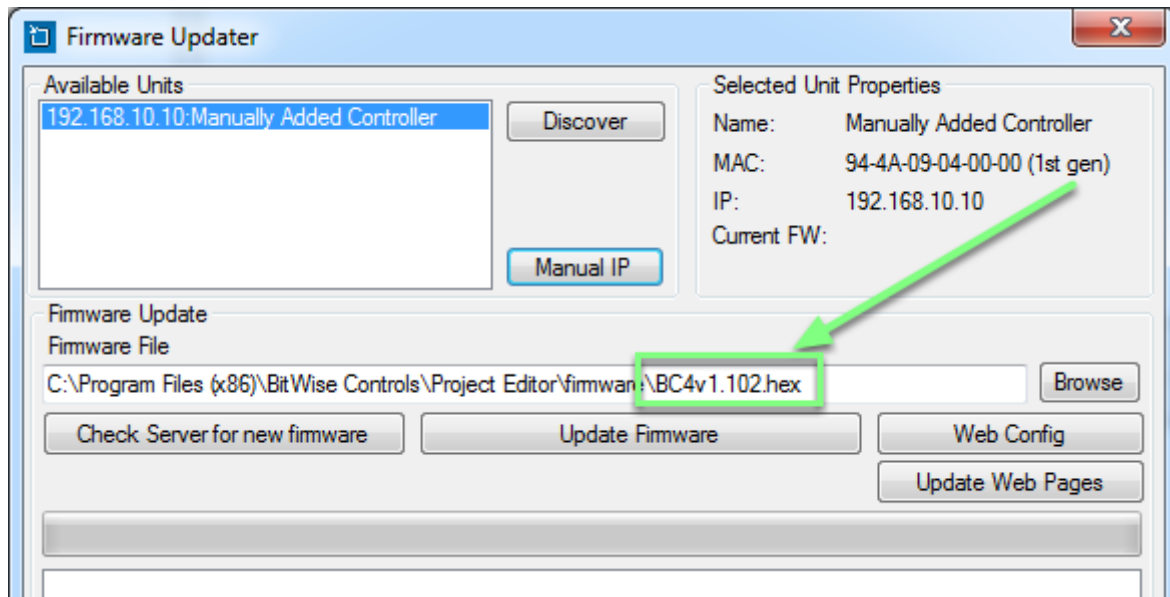
Check Server for New Firmware (optional)



This might be a good time to make sure you have the latest firmware version. [Click Here](#) for more information about firmware updates.

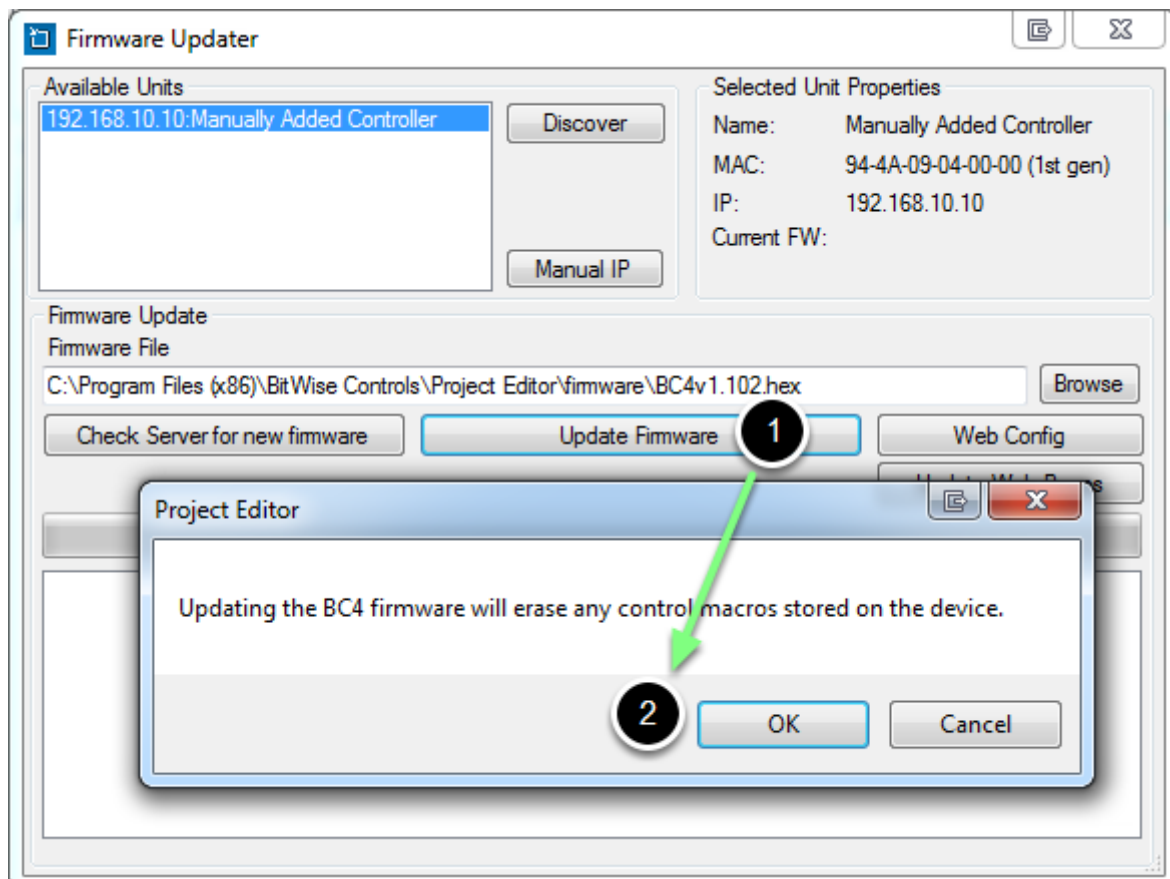
To determine if firmware version is intended for Gen 1 or Gen 2, please go to the Help-> About menu in Project Editor.

Verify Firmware Path



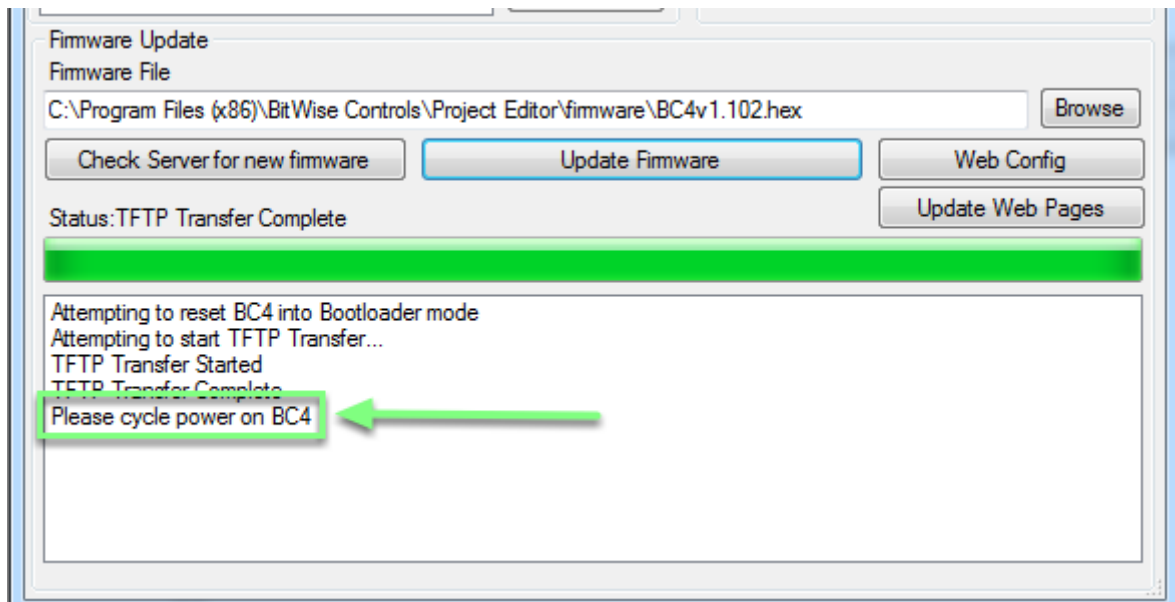
Confirm the path to the firmware file is correct for the firmware version you wish to load. Make certain you do not attempt to load Gen 2 firmware onto a Gen 1 BC4 and vice versa.

Update Firmware



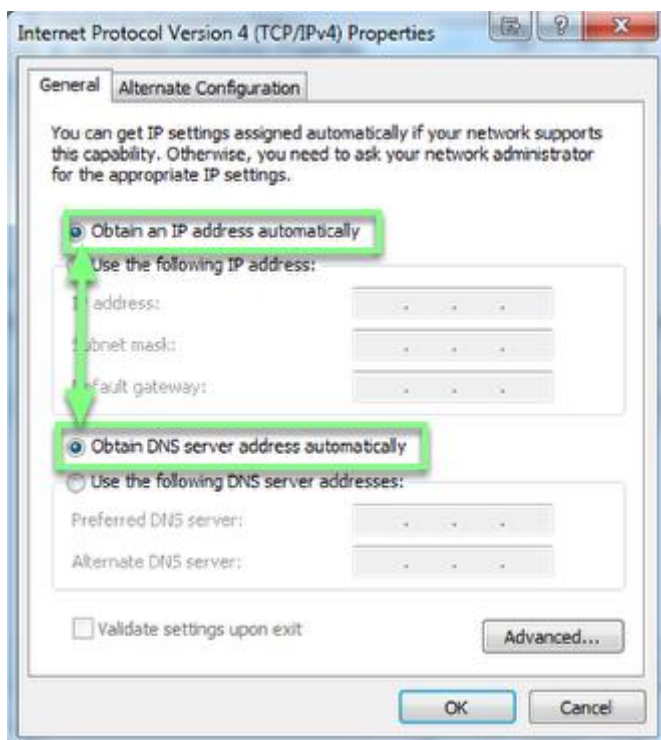
- 1) Click the "Update Firmware" button to load the selected firmware to the manually added BC4
- 2) Click "OK" to acknowledge the alert message.

Close Firmware Updater and Power Cycle the BC4



Click the red X in the upper right corner of the Firmware Updater window and select "No" when prompted to upload project changes (the BC4 will not upload macros when at 192.168.10.10).

Return your PC's Network Settings to Normal



Now that we have recovered the the BC4's firmware, we need to return your computer to the previous network settings. In most cases this will be the top choice for IP address and DNS Server settings. If you took a screen shot before you made changes, now would be the time to reference that.

Return the Computer and BC4 to the Network

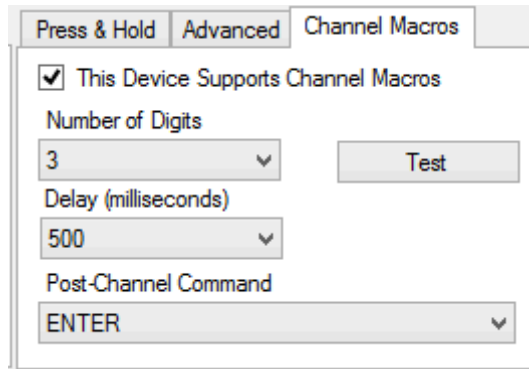
The BC4 can now be returned to the normal network. The firmware update process will have erased any programming previously stored in the BC4, you will need to [upload the unit](#) for macros to work.

NOTE: If the BC4 you are updating was on a firmware version older than v1.07, you will also need to update the internal web pages. If you aren't sure, it won't hurt to [update the web pages](#) anyway.

GUI Channel Macro Tool (Gen 2 Controllers)

All Generation 2 BitWise Controllers support use of the GUI Channel Macro Tool. For the most part, the tool works exactly the same as it did with Gen1 Controllers. Follow the steps below to use the tool in your projects.

Set-Up the IR Device Properties



Press & Hold Advanced Channel Macros

☒ This Device Supports Channel Macros

Number of Digits
3

Delay (milliseconds)
500

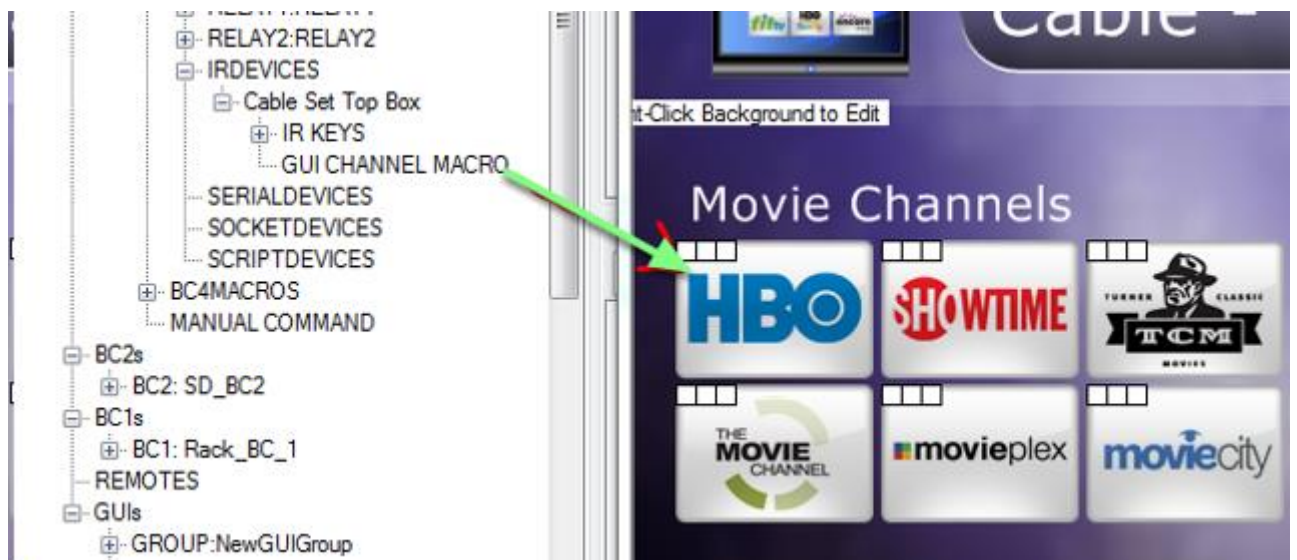
Post-Channel Command
ENTER

Test

1. Tick the box "This Device Supports Channel Macros"
2. Select the maximum number of digits for the device. The macro will zero fill shorter channels (i.e. 9 becomes 009) which may mean you don't need a post-channel command
3. Test your command first without a post command (unless you know it's required)
4. Adjust delay as required
5. Enter Post-Channel command if required and return to step 3 to test

Troubleshooting tip: If your macros work from the Test Button and not from the GUI, try increasing the delay until all commands are sent successfully.

Use the Tool to Add Channel Macros



1. Now that we have set the IR device up to use GUI Channel macros you will notice a new Node under the IR KEYS branch of the device called GUI CHANNEL MACRO. All you need to do now is drag this tool onto your favorite buttons and enter the desired channel number when prompted.
2. Upload your controller and your GUI to test your Favorite Buttons.

IMPORTANT NOTE 1: When using learned commands, be sure to name your digits following the same pattern as the built-in database. DIGIT 1, DIGIT 2, DIGIT 3 and so on... This will ensure the supporting scripts are generated correctly.

IMPORTANT NOTE 2: When using this tool with a Gen 2 BC4 the digit IR codes and post command code must be in use in the project other than the GUI Channel Tool. The codes can be assigned to digit and enter buttons on a GUI page or simply stored in a macro. IF this is not done, the IR data will not become part of the IR table that gets created when uploading the BC4. This is true whether working with the built-in database or learned codes.

Integrating IP Cameras and Servers

To add a compatible IP camera or DVR image to a GUI interface, use the [Web Image feature](#) of the GUI Editor. A Web Image is basically a way to place an image on a GUI page that has an external source. The source can be a static image from a web server or an IP camera.

Compatible Cameras

IP cameras provide their image in several different ways and some are not compatible with the web image object. **The web image object cannot render RTSP, Flash, Java or Active X type camera feeds.** The type of camera feed we are looking for will provide a jpeg or mjpeg image feed.

Camera Paths

Take a look at the following example URLs.

Static Image Feeds (don't click these links, they are for demonstration purposes only)

<http://72.165.31.230/jpg/image.jpg>

<http://donitas.viewnetcam.com:5003/snapshotJPEG?resolution=320x240>

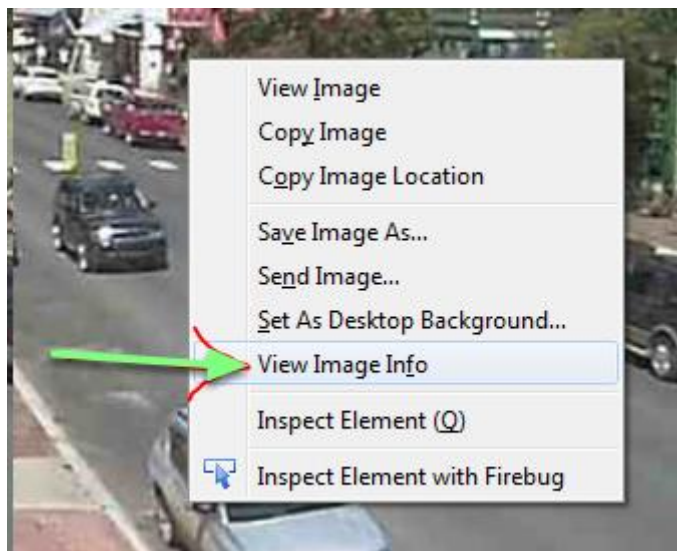
<http://wholebooks.viewnetcam.com:50000/snapshotJPEG?Resolution=640x480&Quality=Standard>

Notice that each URL refers to an image path for a jpeg image (bold text). The parameters in the URL string following the ? are simply being sent to the IP camera to tell it the type of image the browser is requesting.

To find these strings for the cameras you wish to include on a GUI page you first need to view the camera using a web browser. I recommend Firefox for this task due to the format of the View Image Info tool.

Right-Click on the camera image in the web browser and select "View Image Info".

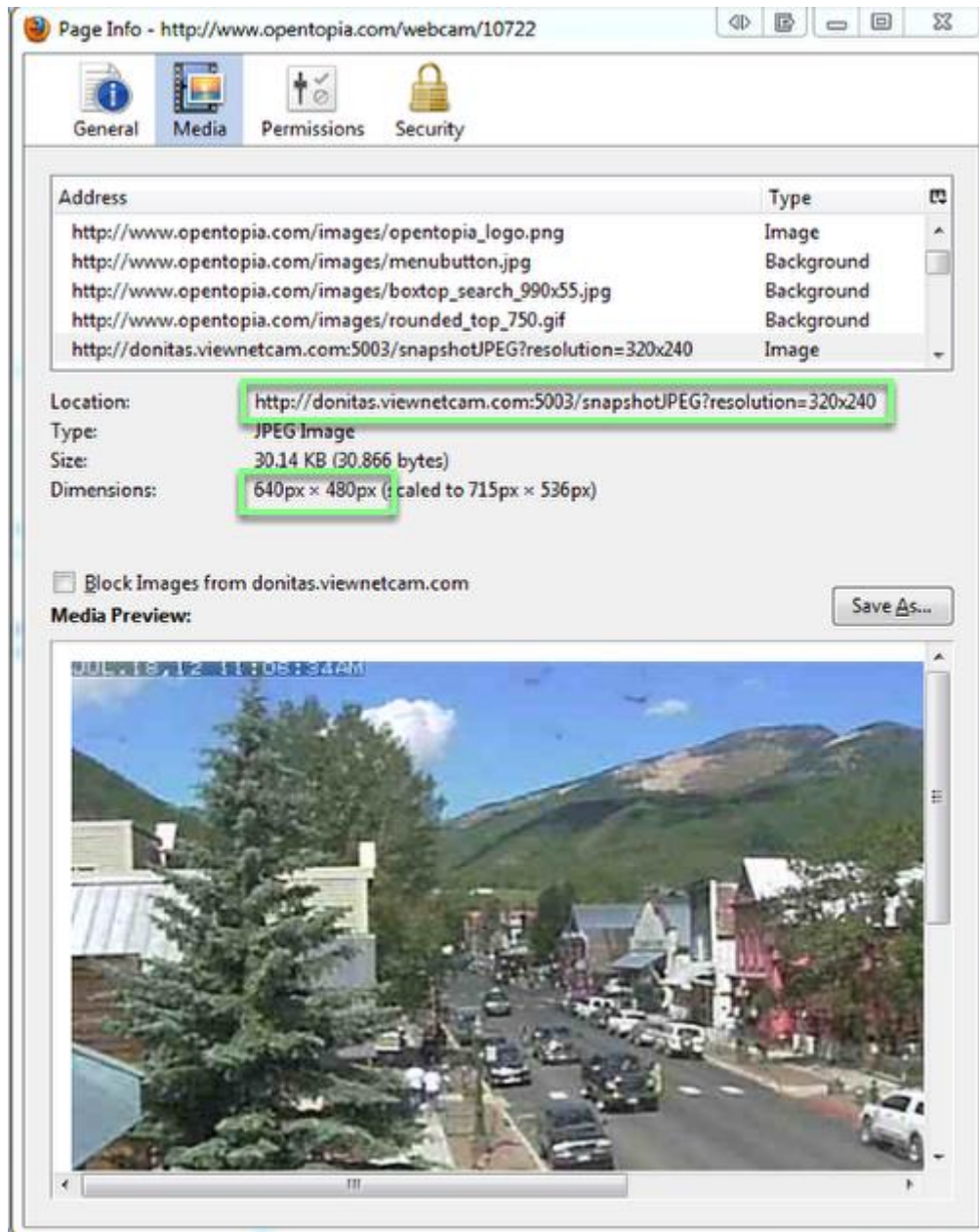
An info window will popup giving you the details you need for the camera feed.



Record Image Properties

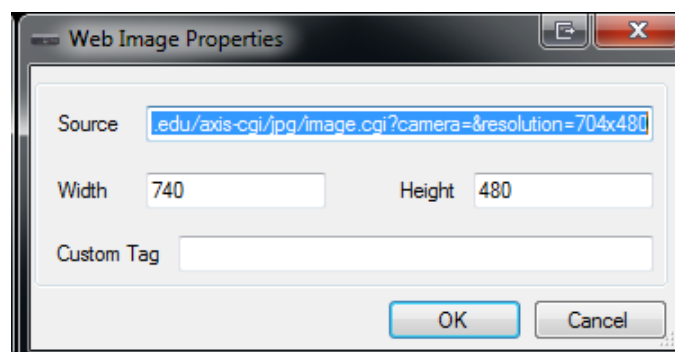
The information you care about are the location and the dimensions. You will need to copy the Location URL to the Source field in the Web Image Properties. You will also want to use the same source and height information to avoid distorting the image.

In all the examples above, the URLs point to a static image. Your camera may also support an mjpeg stream that will automatically refresh at an interval. To get the static image feed to refresh, it is necessary to include a custom GUI script that will refresh the image once per second. Take a look at [this project file](#) for an example of how this custom script would work.



Transfer Information to the Web Image

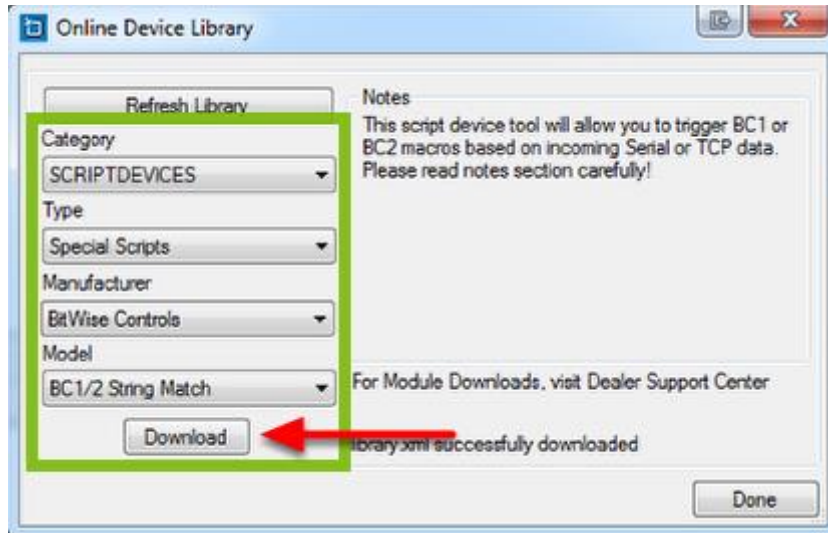
Take the information from the previous step and transfer it to the Web Image properties.



Match Triggers BC1 and BC2

The BC4 has a built-in method for matching incoming serial strings and we are often asked if this can be done with BC1 and BC2 controllers. The answer is yes...but it isn't a built in feature. This article will show you how it can be done using the powerful JavaScript API.

Download the String Match Script Device



The Online Library contains a pre-written Script Device that will look for predefined incoming strings and trigger macros based on the incoming data. This work great if you are sending strings for another system and you have control over the format of the strings. We will start by explaining how this Script Device works and then get into how to modify the JavaScript to work with another system.

Import the Script Device

Open a new or existing project with a BC1 or BC2 controller. Right-click the SCRIPTDEVICES node, then import the file downloaded in the previous step (BC1_BC2_Serial_Match.bscp).

Open the Script Device Properties

Right-click the Script Device, then select properties. Read the Script Device notes to get a general idea how this works.

How does this work anyway?

Script Devices are the device type used when we want to process incoming data (typically to be pushed to the GUI interface). In this case, we are taking the incoming data and using it to execute code locally on the processor. The code you see in a Script Device is present both in the App and in the BC1 or BC2 (BC4s do not run JavaScript). Using BitWise API methods, we can trigger macros on the controller either by Number or by Name.

By Number or by Name?

Macros can be triggered by Macro Number or by Macro Name. Which method is better? Macro names tend to remain the same however the macro number can easily change when macros are reordered in the macro tree. Also, when triggering by name it is important to make sure you name your macros using JavaScript safe names. General rule of thumb is to use letters, numbers and the underscore character but do not start names with a number. [This tool](#) will check for legal names if you like to test something.

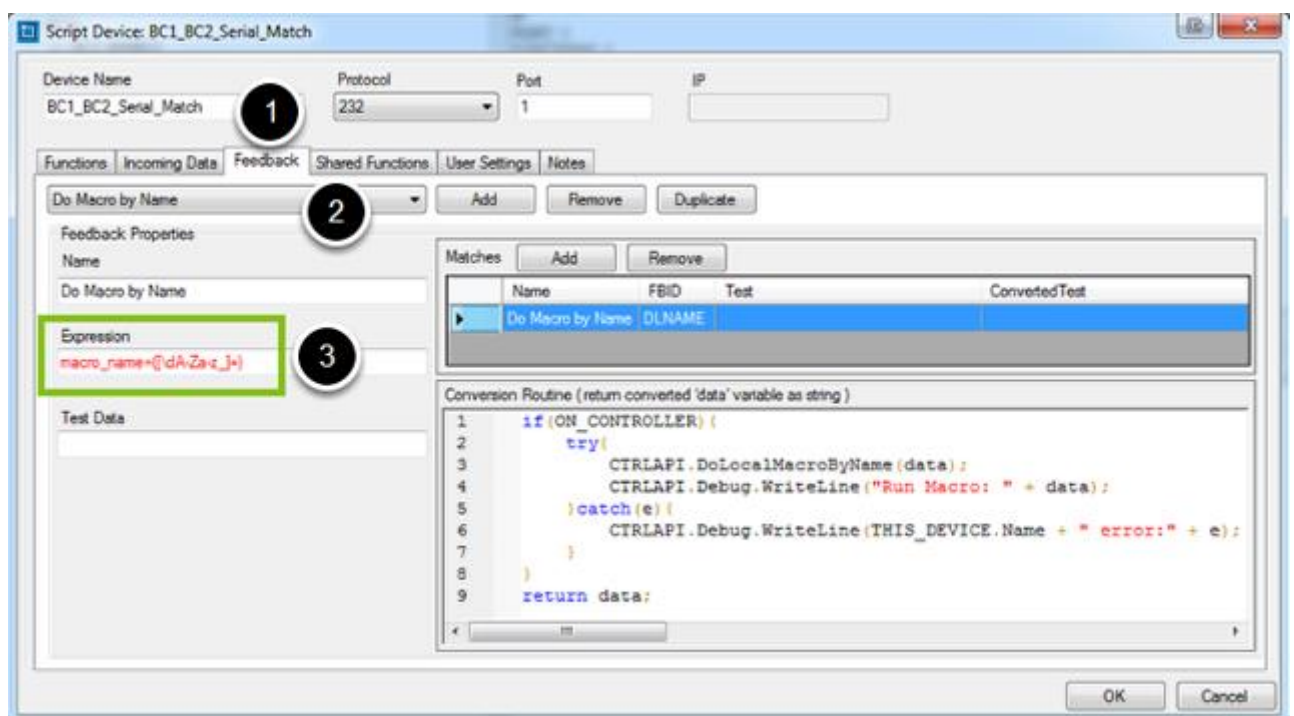
Incoming Data

Data comes into a controller via the RS-232 ports or from a network connection. The Script Device Protocol Setting will determine what stream of incoming data the specific Script Device is listening to. Once we have selected the source of the incoming data, we need to determine how the stream of data gets broken up. Most of the time incoming data will have a Known Terminator since many of the protocols in use were originally used to interface to a device via a terminal. The most common terminator is a Carriage Return, Line Feed pair. In hexadecimal, this is represented by 0D0A (Zero D Zero A...not an Oh). Click on the Incoming Data tab of the Script Device to see the Known Terminator. This device is set to only look for a carriage return 0D. When 0D hex is encountered, the buffer will send the data to the feedback packet handler.

Note: In some rare cases, the incoming data will not have a known terminator. In this case, the option exists to write a custom parse routine but we won't get into this here.

For more information on incoming data and regular expressions, see the article on [Processing Feedback](#).

The Feedback Tab



Now that the incoming data is broken into packets, let's take a look at the Feedback tab to see how it's filtered. Click on Feedback, then select "Do Macro by Name" from the drop down.

Feedback packets are compared to each expression listed in the Feedback tab to see if there is a match. In this example, the expression used is:

```
macro_name=([dA-Za-z_]+)
```

Let's take a quick look at this expression to understand how this works.

For our purposes, we can consider that an expression has two basic parts.

1. The Pattern - The pattern of characters we want to match **macro_name=**
2. Capture Group - The part of the string we want to capture ([dA-Za-z_]+) The character class in the square brackets will allow any digit, letters A-Z (upper or lower case) and the underscore.

For more information on incoming data and regular expressions, see the article on [Processing Feedback](#).

Example:

Let's assume the controller received a string on serial port 1 with the following characters (The Carriage Return that would have triggered the feedback routine gets stripped off before it gets to this point).

macro_name=All_Off

The pattern will match our expression because it contains the literal pattern **macro_name=** and the token pattern in the capture group. The capture group will now store **All_Off** and store it in the first Match and Feedback ID (FBID). If there is code present in the conversion routine for the match it will run with the captured available in a variable named **data**.

Conversion Routine Code: (Text preceded by // are comments)

```
// First, check to see if the code is running on a controller
// If this evaluates to false, we must be on a GUI so skip it
if(ON_CONTROLLER){
    // Try/catch not required but will make any bad data more forgiving
    try{
        // Since data now = "All_Off", the API will trigger the macro
        named "All_Off"
        // This is the API call to trigger a macro by name
        CTRLAPI.DoLocalMacroByName(data);
        // Print to the debug window for verification
        CTRLAPI.Debug.WriteLine("Run Macro: " + data);
    }catch(e){
        // Print the error if it does not work
        CTRLAPI.Debug.WriteLine(THIS_DEVICE.Name + " error:" + e);
    }
}
// Return the original data string to the converted text
// We could still manipulate it if we wanted to
return data;
```

Altering the Script Device

Now that you understand how the process works, you can apply the technique to other Script Devices.