APPLICATION NOTE

Data Logger

HTTP Troubleshooting

HTTPGet(), HTTPPost(), and HTTPPut()

Revision: 03/2025

Table of Contents

1. Introduction

This troubleshooting guide provides detailed instructions for setting up and diagnosing issues with `HTTPGet()`, `HTTPPost()`, and `HTTPPut()` functions in CRBasic. These functions enable Campbell Scientific data loggers to interact with HTTP-based web services, allowing for data retrieval and submission over the internet.

The guide is designed to help users overcome common challenges, including ensuring proper network connectivity, configuring DNS settings, and writing accurate CRBasic code. Whether you are sending a simple HTTP GET request or constructing complex HTTP POST or PUT requests with authentication headers, this document walks you through each step of the process.

2. Verify Internet/network connectivity

Ensure your data logger has an Internet or network connection and can reach the HTTP server where you will be sending your `HTTPGet()`, `HTTPPost()`, or `HTTPPut()`.

To verify connectivity, perform a ping from the data logger to the target IP address or domain using one of these methods:

1. For all Campbell Scientific data loggers, you can use the `PingIP()` instruction in your CRBasic program to check if an address is available. To implement this, declare the `PingResult` variable as `Public` at the top of your program and assign it the result of `PingIP()` within the scan.

   Example:

   ```
   Public PingResult

   PingResult = PingIP ("8.8.8.8",1000)
   ```

2. The CR300 Series data loggers have the added ability to ping from the terminal. To check connectivity for CR300 Series data loggers, connect a computer to the data logger and open the terminal emulator in Campbell Scientific support software, such as ***LoggerNet*** or ***PC400***. In the terminal, press **Enter** until the `CR3XX>` prompt appears. Then, type: ping

x.x.x.x. Replace X.X.X.X with the IP address or name of the HTTP server you are sending data to.

3. Enable **Ping (ICMP) Enabled** in the data logger settings using the ***Device Configuration Utility*** under the **Network Services** tab, then click **Apply**. Connect a laptop to the same network as the data logger and ping the IP address of the data logger over the network from the command prompt to verify the data logger is responding.

   Sending a ping from your data logger to the gateway address your data logger is using will help you verify network connectivity of the data logger. Likewise, performing another ping to an IP address on the Internet, like 8.8.8.8, will help you confirm Internet connectivity.

4. If these methods are successful, and you are going to be sending HTTP data data to a server using a domain name (e.g. myhttpserver.com), be sure to send a test ping to that server address. If the ping is unsuccessful, your data logger may not have a DNS server to resolve the server address to an IP address, or your network connection may not be functioning properly.

   To verify if you're experiencing a DNS issue, refer to the Troubleshooting DNS resolution failures (p. 19) section in this document.

3. Set DNS information if necessary

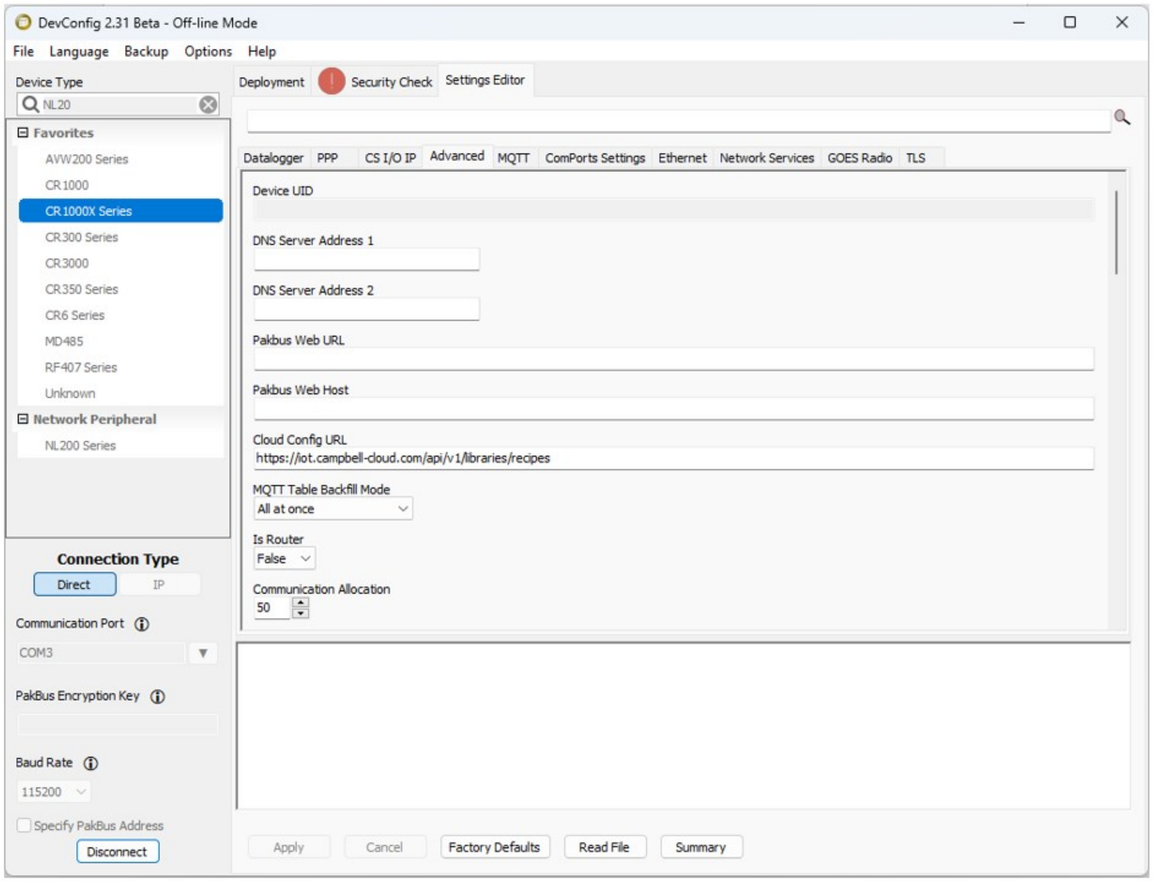
If you can ping the data logger's network gateway and an IP address on the Internet, such as 8.8.8.8 (Google DNS), but cannot ping the server, you may need to specify DNS servers to resolve the server address. You can set these in the data logger using the ***Device Configuration Utility*** under **Settings Editor** > **Advanced**, in the **DNS Server Address** fields.

Settings Editor > **Advanced**

Two common DNS addresses that are often used are:

208.67.222.222 – Open DNS

8.8.8.8 – Google DNS



Device Type: NL20

Cloud Config URL: https://iot.campbell-cloud.com/api/v1/libraries/recipes

MQTT Table Backfill Mode: All at once

Is Router: False

Communication Allocation: 50

Connection Type: Direct | IP

Communication Port: COM3

Baud Rate: 115200

4. Validating your CRBasic code for `HTTPGet()`

NOTE:
If you are using `HTTPPost()` or `HTTPPut()`, skip this section and go to Validating your CRBasic code for HTTPPost() and HTTPPut() (p. 6).

1. Instruction Placement – While you may choose to put the `HTTPGet()` instruction in the main Scan loop, it is more common to put the `HTTPGet()` in a Slow Sequence Scan after the Main Scan:

```
  SlowSequence
  Scan (1,Hr,3,0)

    HTTPResult = HTTPGet("https://posttestserver.dev",Response,httpHeader,7500)

  NextScan
EndProg
```

2. Ensure your `HTTPGet()` instruction includes a result variable to monitor the transaction's success. Declare the `HTTPResult` variable at the top of the program under Public variables:

   ```
   Public HTTPResult As Long
   ```

   Then, prepend it to the `HTTPGet()` instruction with an equal sign, as shown below:

   ```
   HTTPResult = HTTPGet("https://posttestserver.dev",Response,HttpHeader,7500)
   ```

3. Set or Confirm the address in the URI/URL is correct:

   ```
   HTTPResult = HTTPGet("https://posttestserver.dev",Response,HttpHeader,7500)
   ```

4. Ensure a variable has been declared at the top of your program and is used as the `HTTPResponse` parameter. This variable needs to be set as a `Type String` in your `Public` statement and should be large enough to store a full success response or error message from the HTTP server you are querying:

   ```
   Public Response As String * 200

   HTTPResult = HTTPGet("https://posttestserver.dev",Response,HttpHeader,7500)
   ```

5. Set or Confirm the `HTTPHeader` parameter. If your request doesn't require a header, use empty double quotes, e.g.: HTTPResult = HTTPGet ("https://posttestserver.dev", Response,"",7500):

   ```
   HTTPResult = HTTPGet("https://posttestserver.dev",Response,"",7500)
   ```

   If you need to specify a particularly long header, declare a string variable to hold the header in the `Public` variables section of your data logger program. The example below uses the `httpHeader` variable as the header parameter:

```
Public httpHeader As String * 2500
```

```
HTTPResult = HTTPGet("https://posttestserver.dev",Response,HttpHeader,7500)
```

NOTE:
It is generally best to set the header in your CRBasic program just before sending the HTTP request. This is because useful information from the server can be stored there as part of a server's response. That information can be parsed in your data logger program for further use.

For help framing your header, see Basic authentication (p. 12) in this document.

6. Check the `Timeout` parameter. This parameter is optional. The `Timeout` parameter is the amount of time the data logger will wait for a response before considering the connection attempt a failure and incrementing the result parameter from step 1. The time is specified in .01 second intervals. If you have not specified a timeout, then the default of 7500 (75 seconds) is used.

   If your `Timeout` parameter is too short, the `HTTPGet()` instruction may not have enough time to get a response before the data logger stops listening for a response from the server. This is particularly applicable if your server is using a HTTPS or TLS connection, as those will take a little longer. The default of 7500 is generally sufficient for most `HTTPGet()` transactions.

```
HTTPResult = HTTPGet("https://posttestserver.dev",Response,HttpHeader,7500)
```

CAUTION:
Placing an HTTP instruction with a long timeout in the main `Scan` can result in skipped scans. If the timeout exceeds the `Scan` interval, reduce the timeout to avoid skipped scans in case of failure, or place the HTTP instruction in a `SlowSequence`. This prevents the data logger from waiting for a response and missing a scheduled measurement.

5. Validating your CRBasic code for `HTTPPost()` and `HTTPPut()`

What’s the difference between `HTTPPost` and `HTTPPut`?

The HTTP POST method sends data in the request body to a web server, typically for storage. It is commonly used for file uploads and web form submissions. POST helps keep data private and allows the transmission of large amounts of data when needed.

In essence, HTTP Post says, "Here’s the data."

The HTTP PUT method requests the enclosed entity be stored at the supplied URI/URL. If the URI/URL refers to an already existing resource, it modifies it, and if the URI/URL does not point to an existing resource, then the server can create the resource or object with that URI/URL. Unlike HTTPPost(), PUT can create new resources. The data sent represents the complete entity itself. PUT allows multiple resource creations and eliminates the need to check for duplicate submissions.

In essence, HTTP Put says, "Here’s the resource. Create it, or I’m updating the resource you already have."

NOTE:
The parameters for the `HTTPPost()` and `HTTPPut()` instructions in CRBasic are identical. This document uses `HTTPPost()` in its examples.

```
HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwyp8rev73x/post",
ContentsString,HTTPResponse,HTTPHeader,0,0,Min,8,7500)


HTTPResult=HTTPPut ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
ContentsString,HTTPResponse,HTTPHeader,0,0,Min,8,7500)
```

1. Instruction placement – There are reasons you might want to put the `HTTPPost()` or `HTTPPut()` instruction in the main `Scan` loop, but it is more common to put them in a `SlowSequence` scan after the main `Scan`:

NOTE:
You can also put these instructions in a subroutine.

```
Example:

SlowSequence
Scan (1,Hr,3,0)

  HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
  ContentsString,HTTPResponse,HTTPHeader,0,0,Min,8,7500)

NextScan
EndProg
```

2. Ensure the `HTTPPost()` or `HTTPPut()` has a `Result` variable you can monitor to verify the transaction was executed successfully. Declare the `HTTPResult` value at the top of your program in your `Public` variables:

   ```
   Public HTTPResult As Long
   ```

   Then include it on the front of the `HTTPPost()` instruction, followed by an equal sign:

   ```
   HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
   "Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
   ```

   The response codes are listed in this document under step 2 in Verify result and response codes (p. 10).

3. Set or confirm the address in the **URI/URL** is correct:

   ```
   HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwyp8rev73x/post",
   "Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
   ```

4. Define the **Contents** you will send to the HTTP server. You can specify a table name in quotes to send the data from that table, or you can enter a string variable (not in quotes) to include the information you want to Post or Put to the server. You can also send files from the CPU:, CRD:, USB:, or USR: drive by preceding the file name with the drive.

   ```
   HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
   "Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)

   HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
   ContentsString,HTTPResponse,HTTPHeader,0,0,Min,8,7500)

   HTTPResult=HTTPPost
   ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post","CPU:Batt_
   Volt.csv",HTTPResponse,HTTPHeader,+ CHR(13)+ CHR(10) + http_header_content)
   ```

5. Set or Confirm a variable has been declared at the top of your program and is used as the `HTTPResponse` parameter. This variable needs to be set as a `Type String` under `Public` variables and should be large enough to store a full success response or error message from the HTTP server you are querying:

   ```
   Public Response As String * 200
   ```

   ```
   HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
   "Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
   ```

   A quick reference for the most common Response codes is contained in this document in Verify result and response codes (p. 10). For a more complete reference go to Quick reference for HTTPPost(), HTTPPut(), and HTTPGet() errors (p. 20).

6. Set or Confirm the `HTTPHeader` parameter. If your request doesn't require a header, use empty quotes for this parameter. For long headers, declare a string variable in the `Public` variables section of your data logger program, ensuring it is large enough to hold the entire header. The example below uses a string variable named `HTTPHeader` as the header parameter:

   ```
   Public HTTPHeader As String * 2500
   ```

   ```
   HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
   "Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
   ```

   NOTE:
   It is generally best to set the header in your CRBasic program just before sending the HTTP request. This allows useful information from the server's response to be stored in the header, which can then be parsed in your data logger program for further use.

7. If you are streaming data from a data table, you need to specify the `NumRecsStream`, `IntervalStream`, and `UnitsStream` parameters. If you are sending a file (e.g., `USR:myfile.dat`), then you do not need to specify those parameters.

   NOTE:
   The `NumRecsStream`, `IntervalStream`, and `UnitsStream` are only used when streaming data directly from a data table or data table field.

```
HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
"Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
```

```
HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
"Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
```

If you POST or PUT data from a table based on the number of unsent records, ensure you specify the number of records in the `NumRecsStream` field and set the `IntervalStream` value to 0.

If you POST or PUT data from a table based on a time interval, the `NumRecsStream` parameter functions as a `TimeIntoInterval` parameter and must be specified. For example, setting `NumRecsStream` to 0, `IntervalStream` to 60, and `UnitsStream` to Min means the `HTTPPost()` or `HTTPPut()` instruction will execute at the start of each 60-minute interval.

Using a `NumRecsStream` of 0 and a `IntervalStream` of 0 will tell the data logger to send all previously unsent data when the `HTTPPost()` or `HTTPPut()` is called.

Ensure you specify the time units for the `NumRecsStream` and `IntervalStream` parameters in the `UnitsStream` variable. Sec, Min, Hr, and other options are available.

```
HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
"Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
```

8. Specify the `FileOption` parameter. This parameter is only needed if you are streaming a file. It lets you choose the file format of your `HTTPPost()` or `HTTPPut()` request. A number of options are available including Binary, ASCII, XML, and JSON.

> **NOTE:**
> Option 8 is the standard Campbell Scientific data file format created by *LoggerNet*.

```
HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
"Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
```

9. Check the `Timeout` parameter. This parameter is optional and measured in 0.01-second intervals. If not specified, the default value of 7500 (75 seconds) is used.

The `Timeout` parameter determines how long the data logger will wait for a response before marking the connection attempt as a failure and incrementing the result parameter from Step 1.

If the `Timeout` is too short, the `HTTPGet()` instruction may not receive a response before the data logger stops listening, especially when using HTTPS or TLS, which require additional processing time. The default value of 7500 is generally sufficient for most `HTTPGet()` transactions.

```
HTTPResult=HTTPPost ("https://posttestserver.dev/p/4p73smwvyp8rev73x/post",
"Test",HTTPResponse,HTTPHeader,0,0,Min,8,7500)
```

6. Verify result and response codes

1. After confirming all parameters and testing your `HTTPGet()`, `HTTPPost()`, or `HTTPPut()` request, check the Public table for the `HTTPResult` code, which indicates the success of the transaction.
   - A successful request returns a value of 100 or greater, representing the TCP socket used.
   - A return value of 0 indicates a failure.
   - A return value of -2 means the instruction wasn't executed because the data source wasn't ready with new records in time.
2. To find more details, check the `HTTPResponse` parameter. Below are some common abbreviated response codes to provide additional information. For more details and codes, see the Quick reference for HTTPPost(), HTTPPut(), and HTTPGet() errors (p. 20) section of this document.

   200 – HTTP Request was successful.

   400 Bad Request – The server cannot or will not process the request due to an apparent client error. For example: a malformed request syntax, excessive size, or invalid request message framing.

   401 Unauthorized – You haven't authenticated with the server correctly. It's likely a problem in your header. You may need to specify and engage in the correct authentication type. The error message returned will specify the type of authentication you need to use. Types include Basic, Digest, and Bearer. Alternatively, if using Bearer authentication, this could mean the token or session has expired.

403 Forbidden – Assuming you are making the request correctly, you might not have permissions.

404 Not Found – The requested resource could not be found. Check your address to ensure you are using the right one.

7. Starting your header for different authentication types

Many HTTP devices and servers support basic authentication for executing `HTTPGet()`, `HTTPPost()`, or `HTTPPut()` requests. However, many also require additional parameters for proper server authentication. If you receive a 401 error with your request, you must provide credentials such as a username and password, hash, or token to authenticate with the server before sending or receiving data. The 401 error message often provides additional information about the authentication method you should use.

NOTE:
Regardless of the authentication mode, Campbell Scientific data loggers do not support caching with a cookie.

NOTE:
It is generally best to set the header in your CRBasic program just before sending the HTTP request. This allows useful information from the server to be stored as part of a server, which can then be parsed in your data logger program for further use.

Authentication methods are discussed in the following sections:

7.1 Plain text authentication

1. Your data logger sends the username and password in the URI/URL request.
2. The server accepts the request.

Example URI/URL:

http://username:password@http.server.address

7.2 Basic authentication

1. Your data logger connects to the web server and sends its username and password encoded as Base64.
2. The server then processes the request.

Use this format:

```
"Authorization: Basic QWRtaW4=:UGFzc3dvcmQ="
```

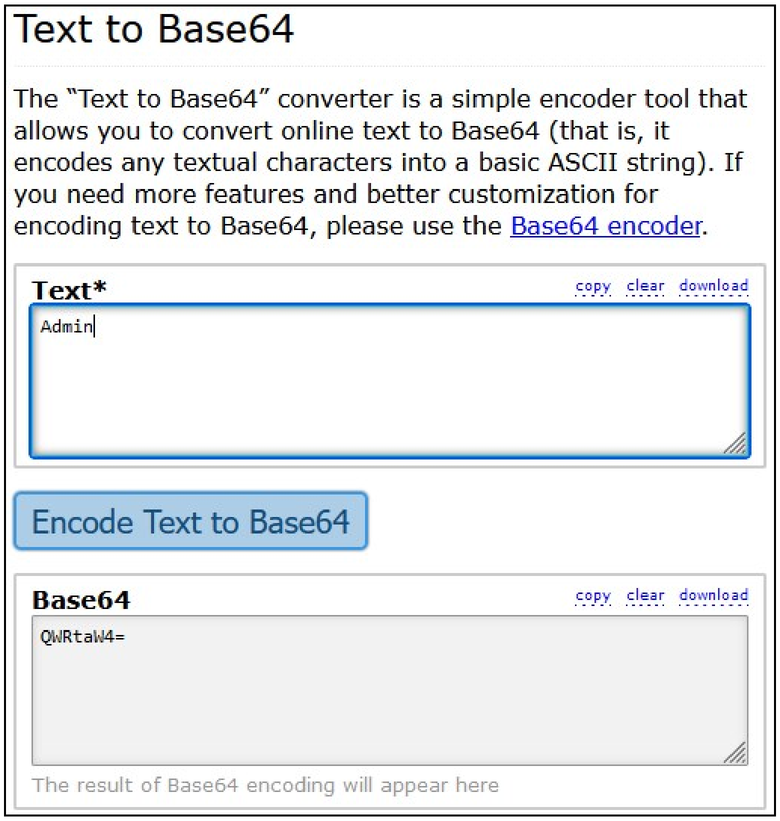
Avoid this format:

```
"Authorization: Basic Admin:Password"
```

To easily convert your username and password into Base64, you can use online tools like Base64 Guru.

https://base64.guru/converter/encode/text

To convert, enter your username or password, then click **Encode**. For example, encoding "Admin" results in `QWRtaW4=`. When using Basic Authentication with Base64, always separate the username and password with a colon (:).



Text to Base64

The "Text to Base64" converter is a simple encoder tool that allows you to convert online text to Base64 (that is, it encodes any textual characters into a basic ASCII string). If you need more features and better customization for encoding text to Base64, please use the Base64 encoder.

Text* copy clear download

Admin

Encode Text to Base64

Base64 copy clear download

QWRtaW4=

The result of Base64 encoding will appear here

7.3 Digest authentication

(See example code in HTTPPost() digest authentication example [p. 26].)

Digest Process Summary:

If you are using this method, start with our sample code and modify it for use with the HTTP server you are communicating with.

1. The client asks for the HTTP resource and returns a 401 error with the authentication realm and a nonce (a "number used once").
2. The data logger reads the error message and parses out the realm and nonce.
3. The data logger creates a hash using the username, password, realm, and nonce.
4. The data logger sends another `HTTPGet()` to the server with the hash in the request's header.
5. The server validates the hash and grants access.

Example Header:

```
"Authorization: Digest xx-XXXXXX"
```

See HTTPPost() digest authentication example (p. 26) for example code.

7.4 Bearer (token) authentication

(See example code in HTTPGet() bearer (token) authentication example [p. 34].)

1. The data logger makes the HTTP Request specifying Bearer in the Header, followed by a token (a hash).
2. The Server accepts the request.

Example Header:

```
"Authorization: Bearer xx-XXXXXX"
```

8. Framing your header in the **HTTPHeader** parameter

Guidelines for framing your header:

- If using a header variable, it must be declared as a string and be large enough to hold the contents.
- The contents of the header variable must be enclosed in quotes.
- Your header must match exactly what the server expects; don’t take shortcuts.
- A long header, like an authorization header, will need to be split up on multiple lines and concatenated together into a single variable.
- If your header needs to include characters, like double quotes or the “-“ minus symbol, they may need to be inserted into the header using the `CHR()` instruction. Here is a reference:

Decimal #	ASCII Char		Decimal #	ASCII Char		Decimal #	ASCII Char
10	LF		42	*		64	@
13	CR		43	+		91	[
32	Space (SP)		44	,		92	\
33	!		45	-		93	]
34	"		46	.		94	^
35	#		47	/		95	_
36	$		58	:		96	`
37	%		59	;		123	{
38	&		60	<		124	|
39	’		61	=		125	}
40	(		62	>		126	~
41	)		63	?			

NOTE:
Most characters, such as ":", can be inserted into your header or any string without using CHR(). However, characters like double quotes (") and the negative sign (–) must be concatenated. The full list of decimal to ASCII characters is also available in the *CRBasic Editor* help under "ASCII Codes and Characters."

Example:

The server requires a string of:

```
"Authorization: Bearer CjBbMUr0-MUQfNzIyRtn"
```

When entered in *CRBasic* it appears as follows:

```
httpHeader = "Authorization: Bearer CjBbMUr0-MUQfNzIyRtn"
```

*Notice the - minus character between CjBbMUr0 and MUQfNzIyRt is blue instead of pink. This indicates a problem in the string.

To correctly set the minus character as a string, use the following code:

```
"Authorization: Bearer CjBbMUr0" & CHR(45) & "MUQfNzIyRtn"
```

NOTE:
The & character is used to concatenate the three pieces together to form the complete string.

9. Managing string length in a header

CRBasic has a 512-character limit per line. If your string exceeds this limit, you must split it into multiple lines and concatenate them.

If your line exceeds 512 characters, you'll encounter this CRBasic compiler error:

```
line 34: Line exceeds max characters of 512.
Error(s) detected in the program. Double-click an error above to navigate to it.
```

Here is an example of splitting a long header string and concatenating it back together:

```
httpHeader1 = "Authorization: Bearer" & CHR(45) & "ONkU5RERFMjBBMUQONURFNzIONkU5
httpHeader2 = httpHeader & "ONkU5RERFMjBBMUQONURFNzIyRTMONjAlMMtYONkU5RERFMjBBMUQ
httpHeader3 = httpHeader & "ONkU5RERFMjBBMUQONURFNzIyRTMONjA1M" & CHR(45) & "ONkU

FullhttpHeader = httpHeader1 & httpHeader2 & httpHeader3
```

10. Testing a request on a generic HTTP server

Testing `HTTPGet()`, `HTTPPost()`, or `HTTPPut()` requests on a generic HTTP test server can help verify your request is formatted correctly. Two commonly used test servers include:

1. https://posttestserver.dev/ – Free, no login currently required.
2. https://pipedream.com/ – Request Bin is accessible with a free account.

> NOTE:
> Post Test Server works for HTTPGet, Post, and Put.

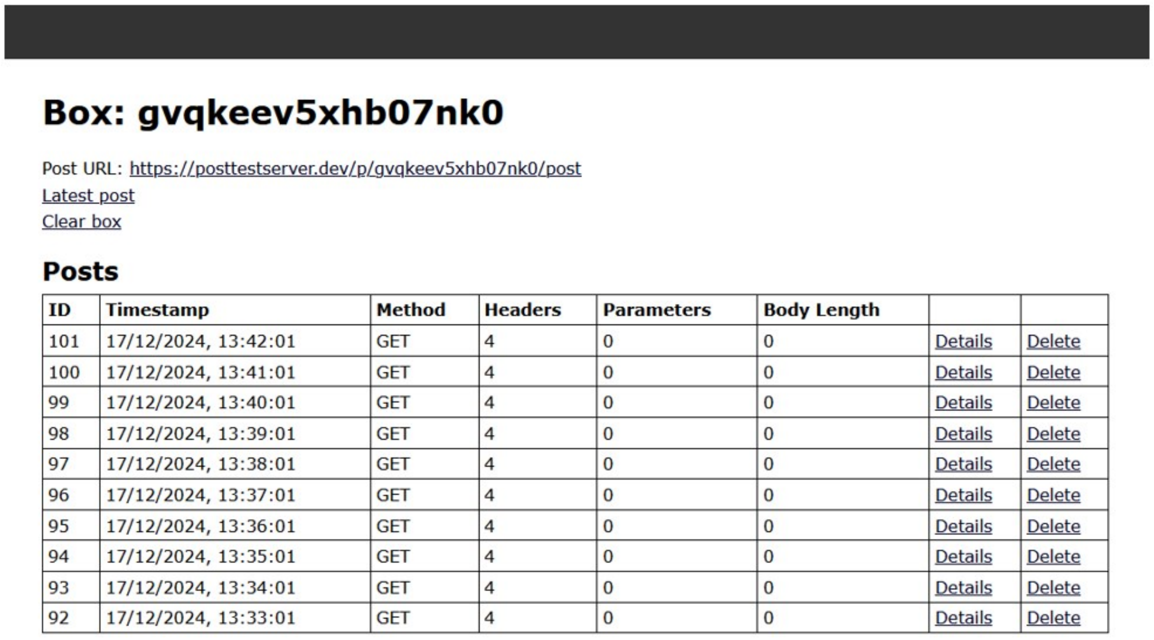
Using Post Test Server:

1. Open the Post Test Server URL
2. Click the **Open Random Box** to generate a test environment.
3. Copy the Post URL to use in your Get, Post, or Put tests in CRBasic.

   > NOTE:
   > Keep the web window open—closing it will delete your test box and results.

4. Test your code and monitor the results in the web window.

   > NOTE:
   > Click Details to view the request headers, parameters, and body of your transaction.



Box: gvqkeev5xhb07nk0

Post URL: https://posttestserver.dev/p/gvqkeev5xhb07nk0/post
Latest post
Clear box

Posts

ID	Timestamp	Method	Headers	Parameters	Body Length		
101	17/12/2024, 13:42:01	GET	4	0	0	Details	Delete
100	17/12/2024, 13:41:01	GET	4	0	0	Details	Delete
99	17/12/2024, 13:40:01	GET	4	0	0	Details	Delete
98	17/12/2024, 13:39:01	GET	4	0	0	Details	Delete
97	17/12/2024, 13:38:01	GET	4	0	0	Details	Delete
96	17/12/2024, 13:37:01	GET	4	0	0	Details	Delete
95	17/12/2024, 13:36:01	GET	4	0	0	Details	Delete
94	17/12/2024, 13:35:01	GET	4	0	0	Details	Delete
93	17/12/2024, 13:34:01	GET	4	0	0	Details	Delete
92	17/12/2024, 13:33:01	GET	4	0	0	Details	Delete

5. Once your test is successful, run your code on the real HTTP server and adjust it based on the server's documentation.

11. Instructions for sniffing `HTTPPost()`, `HTTPPut()`, and `HTTPGet()` from the terminal mode in *Device Configuration Utility*

One way to catch errors such as "Is my header formatted correctly?" and "Does it contain the right type of information?" is to monitor your traffic using a terminal or a terminal emulator.

1. Launch ***Device Configuration Utility*** and connect to your data logger.
2. Click the **Terminal** tab.
3. Type capital letter `W`, and press **Enter**.

4. Enter the number that corresponds to IP Trace (for the CR1000X, it's 23), then press **Enter**.
5. Enter the number that corresponds to HTTP (800), then press **Enter**.
6. You should now see the `press ESC to exit, any other key to renew timeout` message. To create an export of the results, click the **Start Export** button.
7. On the **Choose an export file** window, select a location where you can find the file and give it a name (For example, "mysniff.txt"). Click **Save**.
8. Wait for your `HTTPPost()` instruction to trigger in your data logger program, or trigger it manually. For best results, let the `HTTPPost()` instruction complete or fail twice for redundancy before clicking the **End Export** button. If the `HTTPPost()` instruction only runs once per day at midnight, testing may be difficult without a way to trigger it manually.

12. Instructions for sniffing DNS from the terminal mode in *Device Configuration Utility*

1. Launch ***Device Configuration Utility*** and connect to your data logger.
2. Click the **Terminal** tab.
3. Type capital letter `W`, and press **Enter**.
4. Enter the number that corresponds to IP Trace (for the CR1000X, it's 23), then press **Enter**.
5. Enter the number that corresponds to DNS (e.g., 1000), then press **Enter**.
6. You should now see the `press ESC to exit, any other key to renew timeout` message. To create an export of the results, click the **Start Export** button.
7. On the **Choose an export file** window, select a location where you can find the file and give it a name (For example, "mysniff.txt"). Click **Save**.
8. Capture the results for a few minutes while the data logger attempts the HTTP operation. The data logger must complete a DNS lookup as part of this process. Then, wait a few more minutes. For best results, let the `HTTPPost()` instruction complete or fail twice for redundancy. Once done, click the **End Export** button. If the `HTTPPost()`instruction only runs once per day at midnight, testing may be difficult without a way to trigger it manually.

12.1 Interpreting DNS Sniff file results:

In the resulting file, the following message indicates that the data logger is attempting a DNS lookup on the network::

```
11:21:00.003 sending DNS request ID 65130 for name "google.com" to server 0
```

A message like the one below indicates a successful DNS lookup of the target server:

```
10:27:00.003 dns_lookup: "google.com": found = 142.250.176.14
```

If you see a message like the following, the DNS lookup has failed:

```
11:21:00.028 dns_recv: "afakeURL.com": error 3 in flags
```

NOTE:
Your results will likely include lots of TTL messages, like the following. This is normal.

```
10:26:59.358 dns_check_entry: "google.com": ttl 60
```

13. Troubleshooting DNS resolution failures

If DNS resolution is failing, do the following:

1. Verify the HTTP server address is correct.
2. Verify the data logger is configured with DNS servers.
3. If applicable, verify the data logger is connected to the same private network as the server address to which you are attempting to connect .
4. If applicable, verify the data logger is connected to the Internet.
5. Verify if the data logger is connected to multiple network interfaces and the traffic is being sent to the correct interface. See Instructions for sniffing HTTPPost(), HTTPPut(), and HTTPGet() from the terminal mode in Device Configuration Utility [p. 17].
6. Obtain alternative DNS servers from your local IT department and switch the data logger to those DNS server(s).

7. Work with your local IT department to obtain the IP address associated with the server name. Use this IP address in your CRBasic program as a workaround instead of the server name.

14. Quick reference for `HTTPPost()`, `HTTPPut()`, and `HTTPGet()` errors

2XX Success Messages

This class of code indicates a Success message.

200 OK

Standard response for successful HTTP requests. The actual response will depend on the request method (see https://en.wikipedia.org/wiki/HTTP#Request_methods) used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request, the response will contain an entity describing or containing the result of the action.

201 Created

The request has been fulfilled, resulting in the creation of a new resource.

202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request may or may not be eventually acted upon, and might be disallowed when processing occurs.

3XX Redirection Messages

This class of return indicates the transaction needs to be, or is being, redirected.

300 Multiple Choices

Indicates multiple options for the resource from which the client may choose (via agent-driven content negotiation).

301 Moved Permanently

This and all future requests should be directed to the given URI.

302 Found (Previously "Moved temporarily")

Tells the client to look at (browse to) another URL. The HTTP/1.0 specification required the client to perform a temporary redirect with the same method (the original describing phrase was "Moved Temporarily"), but popular browsers implemented 302 redirects by changing the method to GET. Therefore, HTTP/1.1 added status codes 303 and 307 to distinguish between the two behaviors.

> NOTE:
> A 302 can indicate cookies need to be used in authentication with the server. *This is not supported by Campbell Scientific data loggers.

4XX Client Side Error Messages

This class of status code is intended for situations in which the error seems to have been caused by the client. Except when responding to a HEAD request, the server should include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

400 Bad Request

The server cannot or will not process the request due to an apparent client error (e.g., malformed request syntax, size too large, invalid request message framing, or deceptive request routing).

401 Unauthorized

Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource. This could be with either Basic or Digest Authentication. 401 semantically means "unauthenticated", meaning the user does not have valid authentication credentials for the target resource.

403 Forbidden

The request contained valid data and was understood by the server, but the server is refusing action. This may be due to the user not having the necessary permissions for a resource, needing an account of some sort, or attempting a prohibited action (e.g. creating a duplicate record where only one is allowed). This code is also typically used if the request provided authentication by answering the WWW-Authenticate header field challenge, but the server did not accept that authentication. The request should not be repeated.

404 Not Found

The requested resource could not be found but may be available in the future. Subsequent requests by the client are permissible.

405 Method Not Allowed

A request method is not supported for the requested resource; for example, a GET request on a form that requires data to be presented via POST, or a PUT request on a read-only resource.

406 Not Acceptable

The requested resource is capable of generating only content not acceptable, according to the Accept headers sent in the request.

407 Proxy Authentication Required

The client must first authenticate itself with the proxy.

408 Request Timeout

The server timed out waiting for the request. According to HTTP specifications, "The client did not produce a request within the time the server was prepared to wait. The client MAY repeat the request without modifications at any later time."

409 Conflict

Indicates the request could not be processed because of conflict in the current state of the resource, such as an edit conflict between multiple simultaneous updates.

410 Gone

Indicates the resource requested was previously in use, but it is no longer available and will not be available again. This should be used when a resource has been intentionally removed, and the resource should be purged. Upon receiving a 410 status code, the client should not request the resource in the future. Clients, such as search engines, should remove the resource from their indices. Most use cases do not require clients and search engines to purge the resource, and a "404 Not Found" may be used instead.

411 Length Required

The request did not specify the length of its content, which is required by the requested resource.

412 Precondition Failed

The server does not meet one of the preconditions the requester put on the request header fields.

413 Payload Too Large

The request is larger than the server is willing or able to process. Previously called "Request Entity Too Large".

414 URI Too Long

The URI provided was too long for the server to process. This is often the result of too much data being encoded as a query-string of a GET request, in which case it should be converted to a POST request. Previously called "Request-URI Too Long".

415 Unsupported Media Type

The request entity has a media type that the server or resource does not support. For example, the client uploads an image as image/svg+xml, but the server requires images use a different format.

416 Range Not Satisfiable

The client has asked for a portion of the file (byte serving), but the server cannot supply that portion. For example, if the client asked for a part of the file that lies beyond the end of the file. Previously called "Requested Range Not Satisfiable".

417 Expectation Failed

The server cannot meet the requirements of the Expect request-header field.

421 Misdirected Request

The request was directed at a server that is not able to produce a response (for example, a connection reuse).

422 Unprocessable Content

The request was well-formed (i.e., syntactically correct) but could not be processed.

423 Locked (WebDAV; RFC 4918)

The resource being accessed is locked.

424 Failed Dependency (WebDAV; RFC 4918)

The request failed because it depended on another request and that request failed (e.g., a PROPPATCH).

425 Too Early (RFC 8470)

Indicates the server is unwilling to risk processing a request that might be replayed.

426 Upgrade Required

The client should switch to a different protocol, such as TLS/1.3, in the Upgrade header field.

428 Precondition Required (RFC 6585)

The origin server requires the request to be conditional. Intended to prevent the 'lost update' problem, where a client GETs a resource's state, modifies it, and PUTs it back to the server, but a third party has also modified the state on the server, and results in a conflict.

429 Too Many Requests (RFC 6585)

The user has sent too many requests in a given amount of time. Intended for use with rate-limiting schemes.

431 Request Header Fields Too Large (RFC 6585)

The server is unwilling to process the request because an individual header field, or all the header fields collectively, is too large.

Modified from Wikipedia: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

Appendix A. Example programs

A.1 HTTPPost() basic authentication (Base64) example

CRBasic Example 1: Basic authentication (Base64)

```
Public PanelTempC, Battvolt

Public TestFileSize As Long
Public HTTPHeader As String *100
Public HTTPResponse As String * 180
Public HTTPResult

DataTable (Test,True,-1)
  DataInterval (0,1,Min,10)
  Sample(1,Battvolt,FP2)
  Sample(1,PanelTempC,FP2)
EndTable

BeginProg
  Scan (1,Sec,3,0)
    Battery (Battvolt)
    PanelTemp (PanelTempC,60)

    CallTable Test
  NextScan

  SlowSequence
  Scan(1,Min,0,0)

    If TimeIntoInterval(0,60,Min) Then
      HTTPHeader="Authorization: Basic QWRtaW4=:UGFzc3dvcmQ="
      HTTPResult = HTTPPost("QWRtaW4:QWRtaW4@www.myurl.com/data","Test", _
      HTTPResponse,HTTPHeader)
    EndIf

  NextScan
```

A.2 HTTPPost() digest authentication example

CRBasic Example 2: HTTPPost() digest authentication

```
Const CR = CHR(13)
Const LF = CHR(10)

'HTTPServerIP
Const HTTP_URI = "InsertHTTPURIhere"
Const HTTP_USERNAME = "httpusername"
Const HTTP_PASSWORD = "httppassword"

Dim _ptrDigest As Long

Public i_http As Long

Public PTemp
Public Batt_Volt

Public Auth_Complete As Boolean

Public HTTP_Request_Status As String *50
Public HTTPResponse As String *200

Public HTTP_AuthType As String *10 = "Digest"
Public HTTP_Result As Long = 0
Const HTTP_LineSize As Long = 8
Public HTTP_Lines(HTTP_LineSize) As String *208
Const HTTP_AuthSize As Long = 10
Public HTTP_Auth(HTTP_AuthSize) As String *112
Public HTTP_Header As String *512 = ""
Public HTTP_AuthRealm As String *32 = ""
Public HTTP_AuthNonce As String *64 = ""
Public HTTP_AuthQoP As String *5 = ""

Public MD5_Success(3) As Long
Public MD5_String(3) As String *256
Public Digest_Auth_HA1 As String *48 = ""
Public Digest_Auth_HA2 As String *48 = ""
Public Digest_Auth_Resp As String *48 = ""
Public Digest_HA1_little(4) As Long
Public Digest_HA2_little(4) As Long
Public Digest_Resp_little(4) As Long
Public Digest_HA1(4) As Long
Public Digest_HA2(4) As Long
Public Digest_Resp(4) As Long
Public Digest_NonceCount As Long = 1
Public Digest_nc As String = ""
Public Digest_cnonce_gen(2) As Long
```

CRBasic Example 2: HTTPPost() digest authentication

```
Public Digest_cnonce As String = ""

'Data Tables
'----------------------------
DataTable (Test,1,-1)
  DataInterval (0,2,Min,10)
  Minimum (1,Batt_Volt,FP2,False,False)
  Sample (1,PTemp,FP2)
EndTable

'Subroutines
'----------------------------
'== Generate digest authentication strings
Function _HTTP__DigestStr(_method As String, _uri As String) As Long
  Dim Digest_AuthStr As String *500 = ""
  'Initiate identifier
  Digest_NonceCount += 1
  Digest_nc = FormatLong (Digest_NonceCount,"%8.8x")
  'Generate a unique client identifier
  Digest_cnonce_gen(1) = INT(RND * 2^31)
  Digest_cnonce_gen(2) = INT(RND * 2^31)
  Sprintf(Digest_cnonce,"%8.8x%8.8x",Digest_cnonce_gen(1),Digest_cnonce_gen(2))
  'Generate MD5 checksums for credentials
  'MD5(username:realm:password)
  MD5_String(1) = HTTP_USERNAME+CHR(58)+HTTP_AuthRealm+CHR(58)+HTTP_PASSWORD
  MD5_String(2) = _method+CHR(58)+_uri 'MD5(HTTP_method:digestURI)
  MD5_Success(1) = CheckSum (MD5_String(1),29,0,Digest_HA1_little())
  MD5_Success(2) = CheckSum (MD5_String(2),29,0,Digest_HA2_little())
  MoveBytes(Digest_HA1(),0,Digest_HA1_little(),0,16,4) 'Convert from
  'little-endian to big-endian before printing
  MoveBytes(Digest_HA2(),0,Digest_HA2_little(),0,16,4)
  Sprintf(Digest_Auth_HA1,"%8.8x%8.8x%8.8x%8.8x",Digest_HA1(1),Digest_HA1(2), _
  Digest_HA1(3),Digest_HA1(4))
  Sprintf(Digest_Auth_HA2,"%8.8x%8.8x%8.8x%8.8x",Digest_HA2(1),Digest_HA2(2), _
  Digest_HA2(3),Digest_HA2(4))
  'Generate MD5 checksums for response
  'MD5 (HA1:nonce:nc:cnonce:qop:HA2)
  MD5_String(3) = Digest_Auth_HA1+CHR(58)+HTTP_AuthNonce+CHR(58)+Digest_nc+ _
  CHR(58)+Digest_cnonce+CHR(58)+HTTP_AuthQoP+CHR(58)+Digest_Auth_HA2
  MD5_Success(3) = CheckSum (MD5_String(3),29,0,Digest_Resp_little())
  MoveBytes(Digest_Resp(),0,Digest_Resp_little(),0,16,4)
  Sprintf(Digest_Auth_Resp,"%8.8x%8.8x%8.8x%8.8x",Digest_Resp(1), _
  Digest_Resp(2),Digest_Resp(3),Digest_Resp(4))
  'Build the authorization string
  Digest_AuthStr = "Authorization: Digest"
  Digest_AuthStr += " username="+CHR(34)+HTTP_USERNAME+CHR(34)
  Digest_AuthStr += ", realm="+CHR(34)+HTTP_AuthRealm+CHR(34)
```

CRBasic Example 2: HTTPPost() digest authentication

```
  Digest_AuthStr += ", nonce="+CHR(34)+HTTP_AuthNonce+CHR(34)
  Digest_AuthStr += ", uri="+CHR(34)+_uri+CHR(34)
  Digest_AuthStr += ", algorithm=MD5"
  Digest_AuthStr += ", response="+CHR(34)+Digest_Auth_Resp+CHR(34)
  Digest_AuthStr += ", qop="+CHR(34)+HTTP_AuthQoP+CHR(34)
  Digest_AuthStr += ", nc="+Digest_nc
  Digest_AuthStr += ", cnonce="+CHR(34)+Digest_cnonce+CHR(34)
  Return @Digest_AuthStr
EndFunction
'----------------------------
Sub HTTP__AuthCheck
  Dim i_ac As Long, j_ac As Long
  'Take the WWW-Authenticate request and grab the useful bits
  SplitStr (HTTP_Lines,HTTP_Header, CR + LF ,HTTP_LineSize,5)
  For i_ac = 1 To HTTP_LineSize
    If InStr (1,HTTP_Lines(i_ac),"Digest",2) Then
      HTTP_AuthType = "Digest"
      SplitStr (HTTP_Auth,HTTP_Lines(i_ac),CHR(34),HTTP_AuthSize,5)
      For j_ac = 1 To HTTP_AuthSize
        If InStr (1,HTTP_Auth(j_ac),"realm",2) Then HTTP_AuthRealm = _
        HTTP_Auth(j_ac+1)
        If InStr (1,HTTP_Auth(j_ac),"nonce",2) Then HTTP_AuthNonce = _
        HTTP_Auth(j_ac+1)
        If InStr (1,HTTP_Auth(j_ac),"qop",2) Then HTTP_AuthQoP = _
        HTTP_Auth(j_ac+1)
      Next j_ac
    ElseIf InStr (1,HTTP_Lines(i_ac),"Basic",2)
      HTTP_AuthType = "Basic"
      SplitStr (HTTP_Auth,HTTP_Lines(i_ac),CHR(34),HTTP_AuthSize,5)
      For j_ac = 1 To HTTP_AuthSize
        If InStr (1,HTTP_Auth(j_ac),"realm",2) Then HTTP_AuthRealm = _
        HTTP_Auth(j_ac+1)
      Next j_ac
    EndIf
  Next i_ac
  HTTP_Request_Status = "HTTP AUTH updated"
EndSub

'----------------------------
Sub HTTP__GET
  HTTP_Request_Status = "HTTP (GET) Occuring"
  HTTP_Result = HTTPGet(HTTP_URI,HTTPResponse,HTTP_Header,7500)
  HTTP_Request_Status = "Server Response = "
  If InStr(1,HTTPResponse,"401",2) Then
    HTTP_Request_Status &= "unauthorized"
    Auth_Complete = False
    Digest_NonceCount = 0
```

CRBasic Example 2: HTTPPost() digest authentication

```
    Call HTTP__AuthCheck
  ElseIf InStr(1,HTTPResponse,"200",2) Then
    HTTP_Request_Status &= "success"
    Auth_Complete = True
  ElseIf InStr(1,HTTPResponse,"204",2) Then
    HTTP_Request_Status &= "success, no content"
    Auth_Complete = True
  Else
    HTTP_Request_Status &= "fail, unknown response"
    Auth_Complete = False
  EndIf

EndSub

'Main Prog
'----------------------------
BeginProg

  Scan (2,Min,0,0)
    PanelTemp (PTemp,50)
    Battery (Batt_Volt)

    CallTable Test

  NextScan

  '--------------------------
  'SlowSequence Scan for HTTPGET()
  SlowSequence
  Scan(60,Min,0,0)

    For i_http = 1 To 5
      Select Case i_http
      Case 1 'The first interaction with the http server has no extra headers
        'and will fail, but the Digest Authentication challenge information
        'will be delivered. This is needed to be able to login.
        HTTP_Header = ""
        Call HTTP__GET()
      Case Is > 1 'After the first interaction, use the Digest challenge
        'information to form up the appropriate Authentication headers and
        'try again.
        HTTP_Header = "Connection: keep-alive" + CR + LF
        _ptrDigest = _HTTP__DigestStr("GET",HTTP_URI)
        HTTP_Header += !(String!)_ptrDigest + CR + LF
        HTTP_Header += "Accept: */*" + CR + LF
        Call HTTP__GET()
    EndSelect
```

CRBasic Example 2: HTTPPost() digest authentication

```
      If Auth_Complete = True Then ExitFor
    Next i_http

  NextScan
  EndSequence
EndProg
```

A.3 `HTTPPost()` with an API using an API Key

`HTTPPost()` with an API using an API Key using Notifyre messaging service (your API will interact different; be sure to check your documentation and frame your request accordingly).

CRBasic Example 3: HTTPPost() with an API key

```
Public PTemp, Batt_volt
Public textsendtest As Boolean
Public ContentsString As String *500
Public HTTPResponse As String *200
Public NotifyreURL As String *100
Public campaignname As String *100
Public HTTPHeader As String *200
Public ContentsString1 As String *200
Public ContentsString2 As String *240
Public ContentsString3 As String *200

Public fromnumber As String *15
Public recipient1 As String *15
Public recipient2 As String *15
Public recipient3 As String *15
Public recipient4 As String *15

Public recipientstring1 As String *70
Public recipientstring2 As String *60
Public recipientstring3 As String *60
Public recipientstrings As String *200

Public RecipientNum As Long

Public MessageBody As String *250
Public APIKey As String *100

'Define Data Tables
DataTable (Test,1,-1) 'Set table size to # of records, or -1 to autoallocate.
  DataInterval (0,15,Sec,10)
  Minimum (1,Batt_volt,FP2,False,False)
  Sample (1,PTemp,FP2)
EndTable

'Main Program
BeginProg
  'This is the correct URL. If you make changes, ensure your code matches the
  'cURL example from Notifyre instead of the .Net, Node.js, or PHP.
  'Notifyre API Documentation available here:
  'https://docs.notifyre.com/api/sms-send
  NotifyreURL = "https://api.notifyre.com/sms/send"
  'Your from number needs +1 appended to the front of it
```

CRBasic Example 3: HTTPPost() with an API key

```
fromnumber = "+1XXXXXXXXXX"
'Enter the number of recipients here. The remaining fields will be ignored
RecipientNum = 4
'Your recipients numbers need +1 appended to the front of them. Multiple
'recipients need to be individually enclosed in quotes and separated by a comma.
recipient1 = "+1XXXXXXXXXX"
recipient2 = "+1XXXXXXXXXX"
recipient3 = "+1XXXXXXXXXX"
recipient4 = "+1XXXXXXXXXX"
MessageBody = "Test Text Message! Be sure to check the battery voltage level to
ensure the station is operational after dark."
APIKey = "API Key from Notifyre goes Here"
campaignname = "Enter your Campaign Name you have entered in Notifyre Here"

Scan (1,Sec,0,0)
  PanelTemp (PTemp,15000)
  Battery (Batt_volt)

  CallTable Test
NextScan

SlowSequence
Scan (5,Sec,3,0)

  If textsendtest = -1 Then
    recipientstring1=recipient1
    recipientstrings=recipientstring1
    If RecipientNum = 2 Then
      recipientstring1=recipient1 &CHR(34)&CHR(13)&CHR(10)& "}" &CHR(44) _
      &CHR(13)&CHR(10)&"{" &CHR(13)&CHR(10)&CHR(34)& "type"&CHR(34)&":" _
      &CHR(34)& "mobile_number"&CHR(34)&CHR(44)&CHR(13)&CHR(10)& CHR(34)& _
      "value"&CHR(34)& ":"&CHR(34)& recipient2
      recipientstrings=recipientstring1
    EndIf
    If RecipientNum = 3 Then
      recipientstring1=recipient1 &CHR(34)&CHR(13)&CHR(10)& "}" &CHR(44) _
      &CHR(13)&CHR(10)&"{" &CHR(13)&CHR(10)&CHR(34)& "type"&CHR(34)&":" _
      &CHR(34)& "mobile_number"&CHR(34)&CHR(44)&CHR(13)&CHR(10)& CHR(34)& _
      "value"&CHR(34)&":"&CHR(34)& recipient2
      recipientstring2=CHR(34)&CHR(13)&CHR(10)& "}" &CHR(44) &CHR(13)&CHR(10)& _
      "{" &CHR(13)&CHR(10)&CHR(34)& "type"&CHR(34)&":"&CHR(34)& _
      "mobile_number"&CHR(34)&CHR(44)&CHR(13)&CHR(10)& CHR(34)& "value" _
      &CHR(34)&":"&CHR(34)& recipient3
      recipientstrings=recipientstring1 & recipientstring2
    EndIf
    If RecipientNum = 4 Then
      recipientstring1=recipient1 &CHR(34)&CHR(13)&CHR(10)& "}" &CHR(44) _
```

CRBasic Example 3: HTTPPost() with an API key

```
      &CHR(13)&CHR(10)&"{" &CHR(13)&CHR(10)&CHR(34)& "type"&CHR(34)&":" _
      &CHR(34)& "mobile_number"&CHR(34)&CHR(44)&CHR(13)&CHR(10)& CHR(34)& _
      "value"&CHR(34)&":"&CHR(34)& recipient2
      recipientstring2=CHR(34)&CHR(13)&CHR(10)& "}" &CHR(44) &CHR(13)&CHR(10)& _
      "{" &CHR(13)&CHR(10)&CHR(34)& "type"&CHR(34)&":"&CHR(34)& _
      "mobile_number"&CHR(34)&CHR(44)&CHR(13)&CHR(10)& CHR(34)& "value" _
      &CHR(34)& ":"&CHR(34)& recipient3
      recipientstring3=CHR(34)&CHR(13)&CHR(10)& "}" &CHR(44) &CHR(13)&CHR(10)& _
      "{" &CHR(13)&CHR(10)&CHR(34)& "type"&CHR(34)&":"&CHR(34)& _
      "mobile_number"&CHR(34)&CHR(44)&CHR(13)&CHR(10)& CHR(34)& "value" _
      &CHR(34)&":"&CHR(34)& recipient4
      recipientstrings=recipientstring1 & recipientstring2 & recipientstring3
    EndIf

    ContentsString1 = "{" &CHR(13)&CHR(10)&CHR(34)& "Body"&CHR(34)&":" _
    &CHR(34)& MessageBody &CHR(34)&CHR(44)&CHR(13)&CHR(10)&CHR(34)& _
    "Recipients"&CHR(34)&":["&CHR(13)&CHR(10)&"{" &CHR(13)&CHR(10)&CHR(34)& _
    "type"&CHR(34)&":"&CHR(34)& "mobile_number"&CHR(34)&CHR(44)&CHR(13)&CHR(10) _
    &CHR(34)& "value"&CHR(34)&":"&CHR(34)
    ContentsString2 = recipientstrings &CHR(34)&CHR(13)&CHR(10)& "}" _
    &CHR(13)&CHR(10)&"]"&CHR(44)&CHR(13)&CHR(10)&CHR(34)&"From"&CHR(34)&":" _
    &CHR(34)& fromnumber &CHR(34)&CHR(44)&CHR(13)&CHR(10)&CHR(34)& _
    "AddUnsubscribeLink"&CHR(34)&":"&"false"&CHR(44)&CHR(13)&CHR(10)
    ContentsString3 = CHR(34)&"CampaignName"&CHR(34)& ":" &CHR(34)& _
    campaignname &CHR(34) &CHR(13)&CHR(10)& "}"
    ContentsString = ContentsString1 & ContentsString2 & ContentsString3

    HTTPHeader = "x-api-token: " & APIKey &CHR(13)&CHR(10)& "Content-Type:
    application/json"

    HTTPPost (NotifyreURL,ContentsString,HTTPResponse,HTTPHeader)
    textsendtest = 0
  EndIf

 NextScan
EndProg
```

A.4 HTTPGet() bearer (token) authentication example

CRBasic Example 4: HTTPGet () bearer (token) authentication

```
Public PTemp, Batt_volt
Public HTTPResult As Long

Public httpHead As String * 4048
Public httpHead1 As String * 512
Public httpHead2 As String * 512
Public httpHead3 As String * 512
Public httpHead4 As String * 512

Public Response As String * 200

'Define Data Tables
DataTable (Test,1,-1) 'Set table size to # of records, or -1 to autoallocate.
  DataInterval (0,1,Min,10)
  Minimum (1,Batt_volt,FP2,False,False)
  Sample (1,PTemp,FP2)
EndTable

'Main Program
BeginProg
  Scan (1,Min,0,0)
    PanelTemp (PTemp,15000)
    Battery (Batt_volt)

    CallTable Test
  NextScan

  SlowSequence
  Scan (1,Hr,3,0)
   'Note: Token is simulated for a generic application using HTTPPost()
    httpHead1 = "Authorization: Bearer " & CHR(13) & CHR(10)&
    "rF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgF8Uzm3RGWH2bD
    byE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgR5cCI6ImF0K2p3dCIsIng1dCI6Ik
    5HNmQzaUNoMUYzrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgdCVG9CLXJWVlRCTrF8Uzm3RGWH2bDbyE
    4nqrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgrChQnL52YxgI6MTcyMDcwNrF8rF8Uzm3RGWH2bDbyE4
    nqJPChQnL52YxgL52Yxgy9hcGkubXlrrF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg52YxgrF8Uzm3RGW
    H2bDbyE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg"
    httpHead2 =
    "rF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgrF8Uzm3RGWH2b
    DbyE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgM0EwMUZBQjU1NTRDMTNSpIjoiMU
    E1MjkzQjJFNDhGOTc2M0Y2RrF8Uzm3RGWH2bDbyE4nqJPrF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxgg
    OjrF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxgzm3RGWH2bDbyE4nqJPChQnL52YxgQURTWVNURU0iLCJX
    UklURVNZrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgU1RFTSJdrF8Uzm3RGWH2bDbyE4nqJPChQnL52Y
    xgrcBK3GiekVoY7RW4r0D941pdPAraYhvqO_PYU55wFzso6mUOopz6vG-677"
```

CRBasic Example 4: HTTPGet () bearer (token) authentication

```
    httpHead3 = "rF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg" & CHR(45) &
    "rF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg" & CHR(45) &
    "rF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg" & CHR(45) &
    "rF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgrF8Uzm3RGWH2b
    DbyE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg" & CHR(45) &
    "BMy7A1FHgyo3kIzHTKrF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg5u7Bv803AoCI29c1Y9kNNd5unTf
    u" & CHR(45) & "rF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg" & CHR(45) &
    "rF8Uzm3RGWH2bDbyE4nqJrF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgPChQnL52Yxg"
    httpHead4 = "rF8Uzm3RGWH2bDbyE4nqJPChQnL52YxgrF8Uzm3RGWH2bDbyE4nqJPChQnL52Yxg"
    & CHR(13) & CHR(10)

    httpHead = httpHead1 & httpHead2 & httpHead3 & httpHead4

    HTTPResult = HTTPGet
    ("https://api.exampleURI.com/v2/systems/me?page=1&itemsPerPage=10"
    ,Response,httpHead,7500)

  NextScan
EndProg
```

A.5 HTTPPut() bearer (token) authentication example for Microsoft Azure Blob Storage

CRBasic Example 5: HTTPPut() bearer for Microsoft Azure

```
Public PanelTempC, Battvolt
Public http_post_tx
Public TestFileSize As Long
Public HTTP_RESPONSE As String * 150

Public OutStat
Public LastFileName As String *30

DataTable (Test,True,-1)
  DataInterval (0,1,Min,10)
  TableFile ("USR:TableName",8,-1,0,24,Hr,OutStat,LastFileName)
  Sample(1,Battvolt,FP2)
  Sample(1,PanelTempC,FP2)
EndTable

BeginProg
  Scan (1,Sec,3,0)
    Battery (Battvolt)
    PanelTemp (PanelTempC,60)
    CallTable Test
  NextScan

  SlowSequence
  Scan(1,Min,0,0)

    If TimeIntoInterval(0,60,Min) Then
      TestFileSize = FileSize(LastFileName)
      'The example token details with Azure are not valid. You'll need to
      'substitute real ones.
      http_post_tx = HTTPPut
      ("https://azureserveraddress.blob.core.windows.net/subdirectory/" +
      LastFileName + "?sv=2024-11-09&si=nccospublicstor-subdirectory-
      user.user&sr=c&sig=dFspqhBxwlWYthpe3Ko7Y2w464gaY8d4Qv%2CojBJlfbW%3D"
      ,LastFileName,HTTP_RESPONSE,"x-ms-blob-type: BlockBlob" & CHR(13) & CHR(10) &
      "Content-Type: text/plain" & CHR(13) & CHR(10) & "Content-Length: " +
      TestFileSize & CHR(13) & CHR(10),0,0,Min,8)
      TCPClose (http_post_tx)
    EndIf

  NextScan
```

Global Sales and Support Network

A worldwide network to help meet your needs

Campbell Scientific Regional Offices

Australia

Location:	Garbutt, QLD Australia
Phone:	61.7.4401.7700
Email:	info@campbellsci.com.au
Website:	www.campbellsci.com.au

Brazil

Location:	São Paulo, SP Brazil
Phone:	11.3732.3399
Email:	vendas@campbellsci.com.br
Website:	www.campbellsci.com.br

Canada

Location:	Edmonton, AB Canada
Phone:	780.454.2505
Email:	dataloggers@campbellsci.ca
Website:	www.campbellsci.ca

China

Location:	Beijing, P. R. China
Phone:	86.10.6561.0080
Email:	info@campbellsci.com.cn
Website:	www.campbellsci.com.cn

Costa Rica

Location:	San Pedro, Costa Rica
Phone:	506.2280.1564
Email:	info@campbellsci.cc
Website:	www.campbellsci.cc

France

Location:	Montrouge, France
Phone:	0033.0.1.56.45.15.20
Email:	info@campbellsci.fr
Website:	www.campbellsci.fr

Germany

Location:	Bremen, Germany
Phone:	49.0.421.460974.0
Email:	info@campbellsci.de
Website:	www.campbellsci.de

India

Location:	New Delhi, DL India
Phone:	91.11.46500481.482
Email:	info@campbellsci.in
Website:	www.campbellsci.in

Japan

Location:	Kawagishi, Toda City, Japan
Phone:	048.400.5001
Email:	jp-info@campbellsci.com
Website:	www.campbellsci.co.jp

South Africa

Location:	Stellenbosch, South Africa
Phone:	27.21.8809960
Email:	sales@campbellsci.co.za
Website:	www.campbellsci.co.za

Spain

Location:	Barcelona, Spain
Phone:	34.93.2323938
Email:	info@campbellsci.es
Website:	www.campbellsci.es

Thailand

Location:	Bangkok, Thailand
Phone:	66.2.719.3399
Email:	info@campbellsci.asia
Website:	www.campbellsci.asia

UK

Location:	Shepshed, Loughborough, UK
Phone:	44.0.1509.601141
Email:	sales@campbellsci.co.uk
Website:	www.campbellsci.co.uk

USA

Location:	Logan, UT USA
Phone:	435.227.9120
Email:	info@campbellsci.com
Website:	www.campbellsci.com