**CC3100-F5529-email-via-IFTTT**

V1.0: May 17, 2023

**L Hinz Group LLC**

Tables of Contents

[2 Micro Application Status via Email Extremely Valuable 3](#_Toc135236305)

[3 TI CC3100 Email example as of 1-28-2023 No Longer Works 3](#_Toc135236306)

[4 Working IFTTT “If this, then, that” Send Email Solution 3](#_Toc135236307)

[4.1 IFTTT Service 3](#_Toc135236308)

[4.2 IFTTT http PUT call 4](#_Toc135236309)

[4.3 IFTT Email Received example 4](#_Toc135236310)

[5 IFTTT: “If this, then that” Step by Step set up 4](#_Toc135236311)

[5.1 Example Arduino Send Email via Ethernet Shield 5](#_Toc135236312)

[5.2 Creating an Account 5](#_Toc135236313)

[5.3 Creating an Applet 6](#_Toc135236314)

[5.4 Select Trigger 7](#_Toc135236315)

[5.5 Obtaining the Private Key 13](#_Toc135236316)

[5.6 Testing the Applet 15](#_Toc135236317)

[5.6.1 To trigger an Event with an arbitrary JSON payload 16](#_Toc135236318)

[5.6.2 To trigger an Event with 3 JSON values 16](#_Toc135236319)

[5.6.3 To query a web service 17](#_Toc135236320)

[5.7 To Test 17](#_Toc135236321)

[6 TI HTTP Client Example Test Out 17](#_Toc135236322)

[6.1 TI http\_client example with IFTTT PUT 19](#_Toc135236323)

[7 Merging TI HTTP example back into My Application Attic Fan program with Get Time and Email WiFi code 20](#_Toc135236324)

[7.1 emailAndGetTime.c: Merged GetTime, Send Email, and IFTTT Send Email code 20](#_Toc135236325)

[7.2 Properties 20](#_Toc135236326)

[7.2.1 Resource Linked Resources 20](#_Toc135236327)

[7.2.2 CCS Build Dependencies 21](#_Toc135236328)

[7.2.3 CCS Build MSP430 Compiler Include Options 21](#_Toc135236329)

[7.2.4 CCS Build MSP430 Compiler Predefined Symbols 22](#_Toc135236330)

[7.2.5 Linker File Search Path 23](#_Toc135236331)

[7.3 Config.h merges 23](#_Toc135236332)

[7.4 http\_lib, httpcli.c Changes 24](#_Toc135236333)

[7.5 Function call to send an email via IFTTT 24](#_Toc135236334)

[7.5.1 Tera Term Log 26](#_Toc135236335)

[7.5.2 Webhooks IFTTT Email received 26](#_Toc135236336)

[7.6 1st Working IFTTT on F5529 GetTime and SendEmail via IFTTT 27](#_Toc135236337)

[8 Full emailAndGetTime code 28](#_Toc135236338)

[8.1 Data for Email into Two 64 byte Strings 28](#_Toc135236339)

[8.2 emailAndGetTime.c 35](#_Toc135236340)

[8.3 emailAndGetTime.h 105](#_Toc135236341)

[9 The End 108](#_Toc135236342)

# Micro Application Status via Email Extremely Valuable

The ability to get Micro Application data values /status several times a day, or an email when an alarm or special event has occurred, is extremely valuable for both **Product Development Optimization** and **On going in Service Data**. The big benefit of email is that the data can be received on a Smartphone or PC any place in the world.

We find 128 bytes can send the values / status of a lot of key parameters. I am not sure if there is an IFTTT character limit, it is mainly a micro limit of only 1K of RAM and 128K of FLASH. The micro does not need to send long Parameter / Measurement variable names. One or two characters is plenty as one can soon quickly read it and mentally fill in the words represented by one or two characters.

Example values of the data from my applications

1. BV6.2 is battery voltage is 6.8 volts
2. AT1056382 is Attic temperature last 24 hrs is: Max 105 Deg F, Min 63 Deg F, Current is 82 Deg F
3. Fc3h8.3S1149 is Attic Fan cycles 3, 8.3 hours ON, Fan last Started at 11:49 AM
4. Ps15.3r9.5 is Pool pump: Starting amps 15.3 amps, Running is 9.5 amps

# TI CC3100 Email example as of 1-28-2023 No Longer Works

TI CC3100 **“Send Email”** has worked perfectly for 8 years from 2014 until January 27, 2023 using Gmail as the server to send the email. In mid-2022 Gmail and other mail services increased security to require **AP passwords**. After implementing AP Password, “CC3100 Send Email” again worked. In January 2023 GMAIL, Outlook.com, AOL.com all made some type of security enhancement to block less secure IOT micro’s from sending Emails.

For over 3 months we have had near daily dialog on the TI E2E form **“CC3100SDK: CC3100 SDK-Ver-1.3.1 Email No Longer Works”** to see if we could help **TI fix THEIR CC3100 Send Email example.** (My post typo should be Ver 1.3.0 but can’t correct it.) TI E2E people have been very pleasant to work with, but no one in TI have actually taken a CC3100BOOST and a F5529 or FR6989 micro and tried to fix the TI send email example. It would still be nice if some day this was fixed, but for now this is “a waste of time and resources exercise in futility”!

We now have a working “Send Email” alternative solution using the “IFTTT” service. We are pleased to share this CC3100 – F5529 solution with other TI customers / users. If/when we get a working solution for the CC3100-FR6989 we will be pleased to share that as well.

# Working IFTTT “If this, then, that” Send Email Solution

## IFTTT Service

**IFTTT means ‘If this, then that.’** It is an open-source service that gives the user the freedom to program a response to an event according to their likes. One can create an applet which are chains of conditional statements by a combination of several app services and add triggering parameters.

The IFTTT service is free to use, but you will have to create an account. First go to the following website: <https://ifttt.com/>

A detailed step by step guide is outlined below on how to set up the IFTTT service, so that when an http PUT call is made to **“maker.ifttt.com”,** it will send an email to the email address defined, and with the email body message of the characters defined for the various values.

http://maker.ifttt.com/trigger/SendEmailV2/with/key/ xxxxMyKEYxxxxx?value1=74.5

## IFTTT http PUT call

I have created a file called **“emailAndGetTime.c”** and **“emailAndGetTime.h”** that is 95% all TI CC3100 merged examples of 1) GetTime, 2) Send Email, 3) http\_client, and added my function calls from main.c to start these functions.

A single http call with an attached very long URI is all that is required. The Host name is hard a coded for my use, and I defined 214 character string for the URI.

**#define** HOST\_NAME "maker.ifttt.com"

**char** PUT\_REQUEST\_URI[214] = "/trigger/SendEmailV2/with/key/ xxxxxxMyIFTTTKeyxxxxxx?value1=LHProd15CharNam&value2=1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567-ZX";

**The URI contains multiple elements**

1. **Trigger** is SendEmail2 so it calls the IFTTT service I created called “SendEmail2”
2. **Key** is the unique user/password key you get when setting up your IFTTT service
3. **Value 1**: is the name of the event that is included in the Subject line of the Email from IFTTT. My code overwrites this with each applications specific product name.
4. **Value2**: is two 64 byte data strings from my application. When SendIFTTTemail function is called the code overwrites all those 0123456789’s with the values of the data from my application. For example:
   1. BV6.2 is battery voltage is 6.8 volts
   2. AT1056382 is Attic temperature last 24 hrs is: Max 105 Deg F, Min 63 Deg F, Current is 82 Deg F
   3. Fc3h8.3S1149 is Attic Fan cycles 3, hours on 8.3, Started 11:49 AM
   4. Ps15.3r9.5 is Pool pump starting amps 15.3 amps, running is 9.5 amps

We find 128 bytes can send the values / status of a lot of key parameters. I am not sure if there is an IFTTT limit, it is mainly a micro limit of only 1K of RAM and 128K of FLASH.

## IFTT Email Received example

**From:** Webhooks via IFTTT <action@ifttt.com>   
**Sent:** Sunday, May 14, 2023 12:57 PM  
**To:** my email address  
**Subject:** The event named "LwGrAtticFnHVAC" occurred on the Maker Webhooks service SendEmailV2

What: SendEmailV2  
When: May 14, 2023 at 12:56PM  
Extra Data: LAtFnHVA-M05D14-T1256-S791R712-CD015-F647-P00-G08m38-G08478c00x-A08376-E07973-I7774-R08973-F0t000S1145-H07648c15hy000t017e84-08-ZX

# IFTTT: “If this, then that” Step by Step set up

**IFTTT means ‘If this, then that.’** It is an open-source service that gives the user the freedom to program a response to an event according to their likes. We can create an applet which are chains of conditional statements by a combination of several app services and add triggering parameters.

The IFTTT service is free to use, but you will have to create an account. First go to the following website: <https://ifttt.com/> Click on the ‘Get Started’ button.

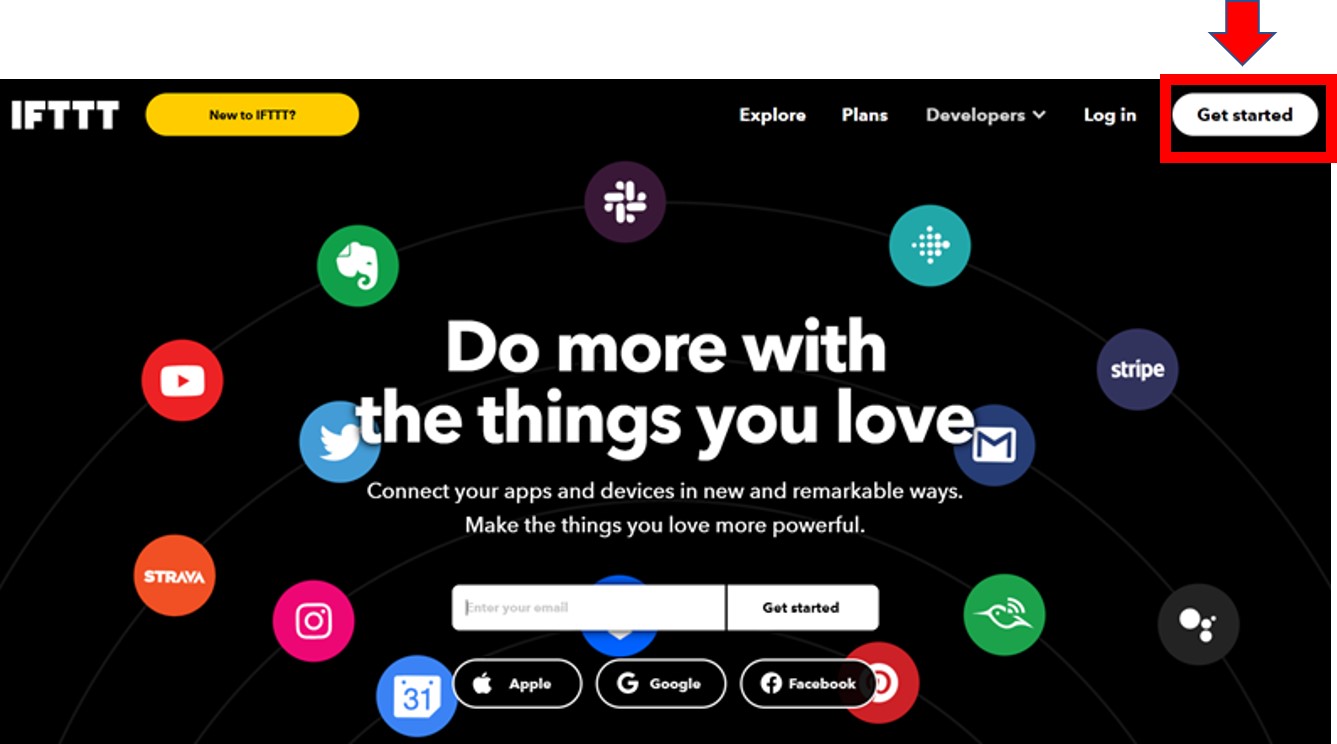
## Example Arduino Send Email via Ethernet Shield

This link provides a step by step process how to set up you IFTTT account and service to send an email. The same steps can be used for your MSP based TI micros.

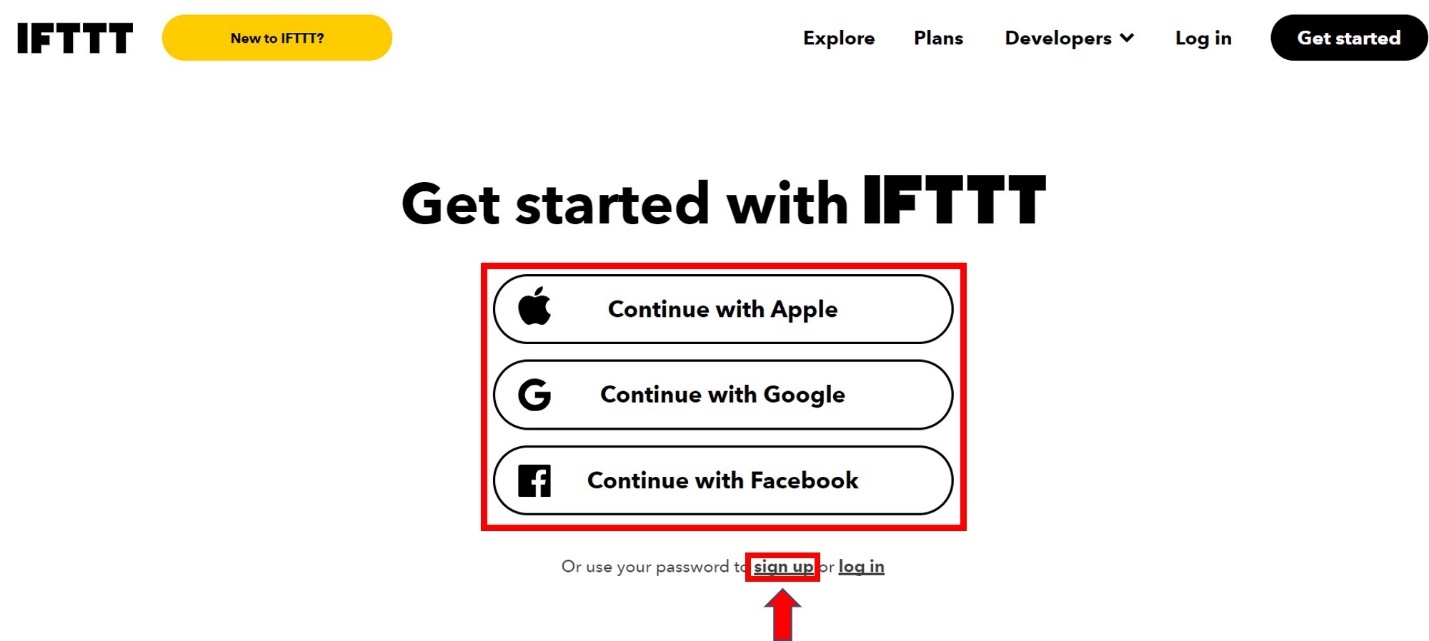
[Arduino Send Email through IFTTT using Ethernet Shield (microcontrollerslab.com)](https://microcontrollerslab.com/arduino-send-email-ifttt-ethernet-shield/)

## Creating an Account

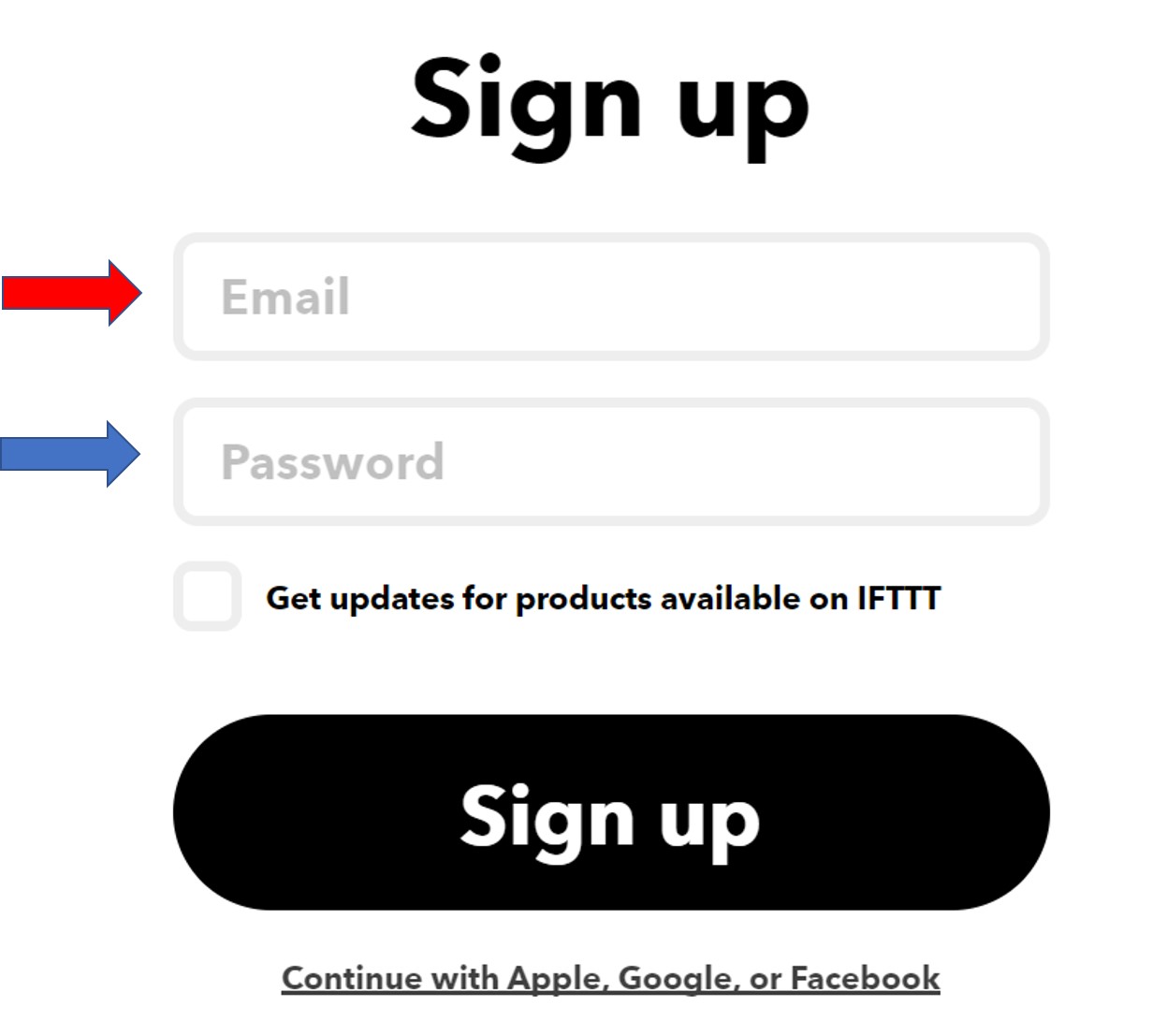
Although the IFTTT service is free to use, we will have to create an account. First go to the following website: <https://ifttt.com/>  
The following window will appear. Click on the ‘Get Started’ button.



The following window will appear. You can select any one from these three options (Apple, Google or Facebook) to connect. Or you can simply ‘sign up’ with your own given email. We will be following this scheme.

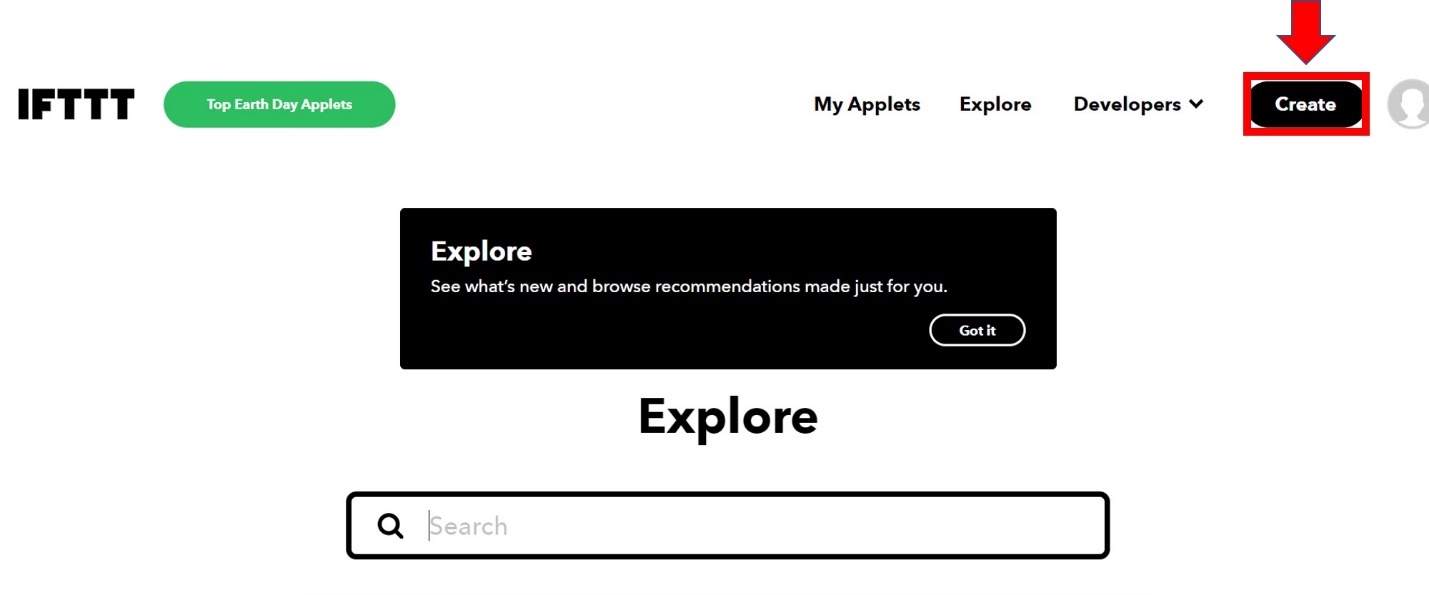


Click the ‘sign up’ tag. You will see the following window pop up. Enter your email address and password to start working in IFTTT. This whole process is free of cost for the first three applets.

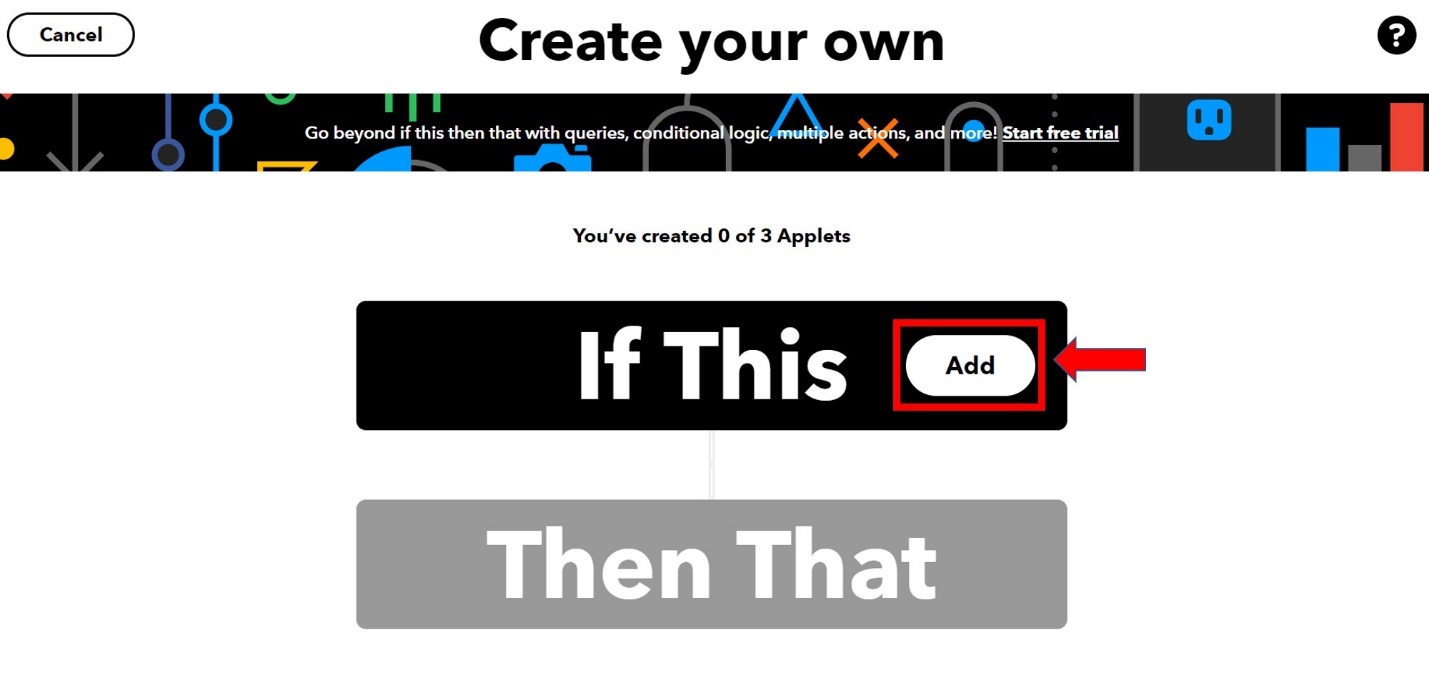


## Creating an Applet

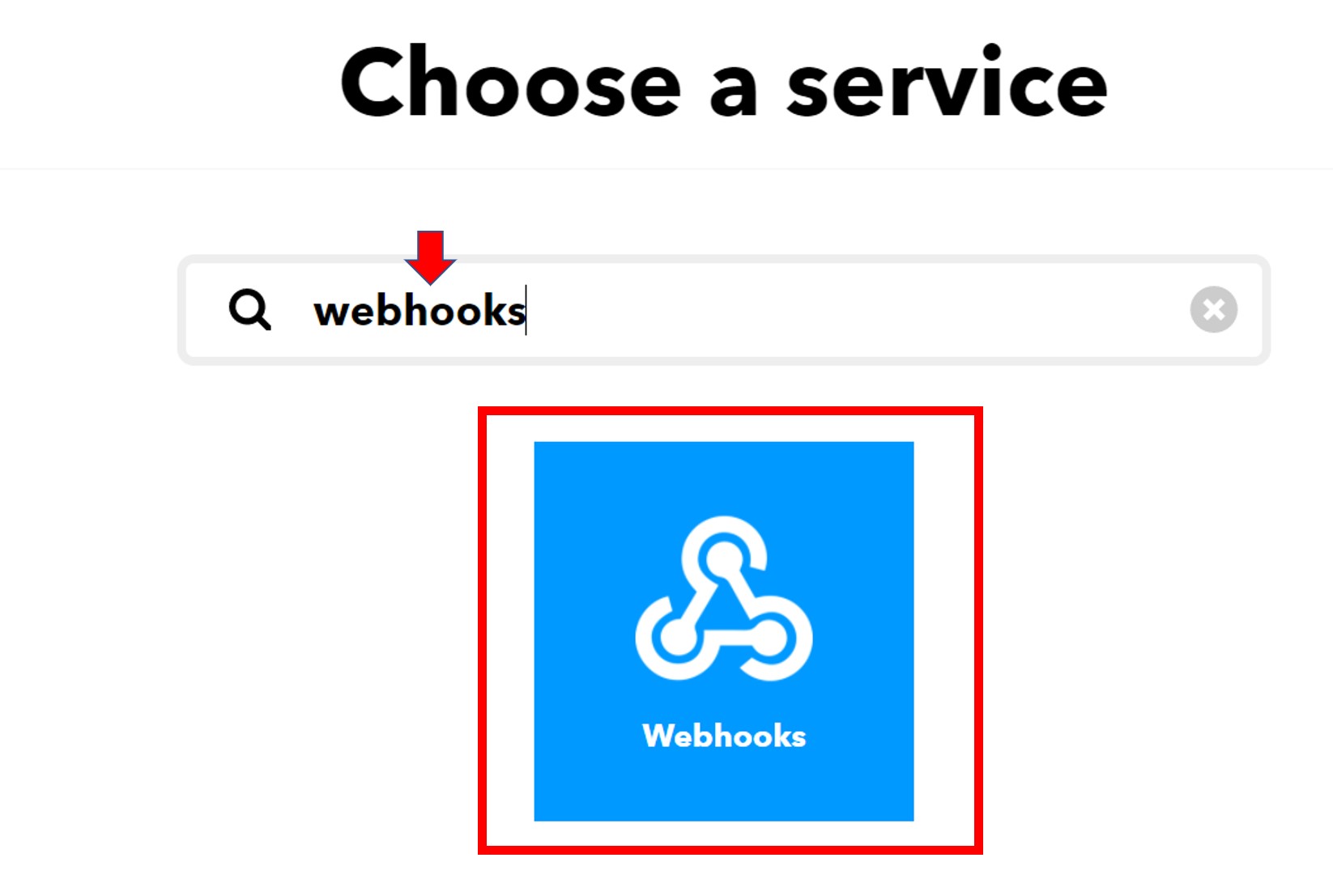
After you have created your account, we will be directed to the page where we will create our applet. Click on ‘Create.’



The following window opens up. Click the following Add button in the ‘If This’ section.

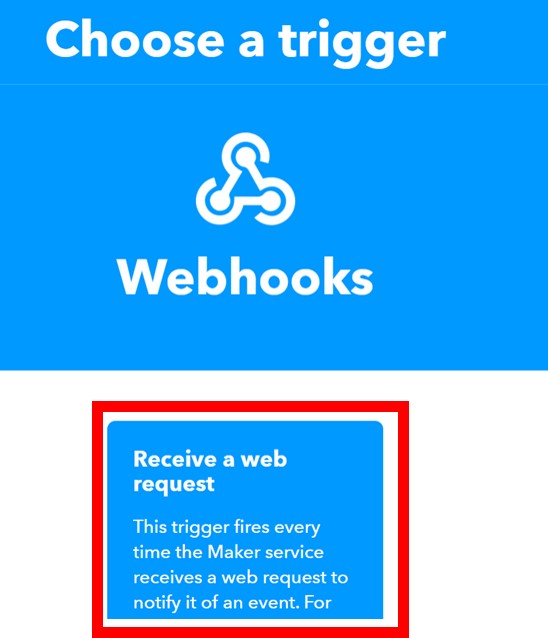


Another page will open in which we will have to choose our service. There is a lot of options to choose from. Write down **‘webhooks’** in the search option and its icon will appear:

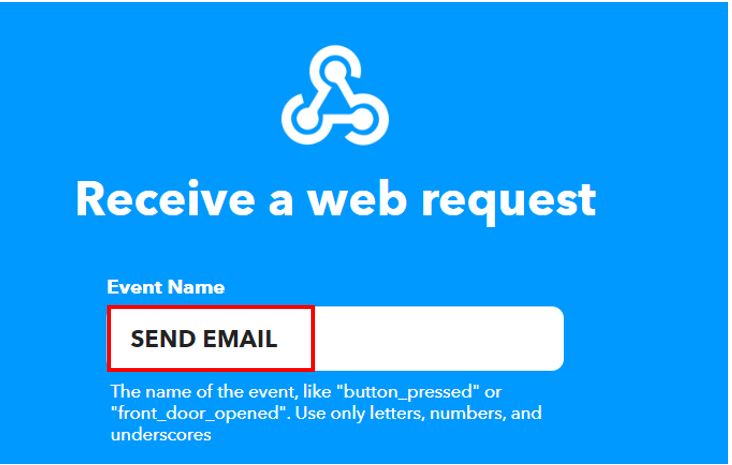


## Select Trigger

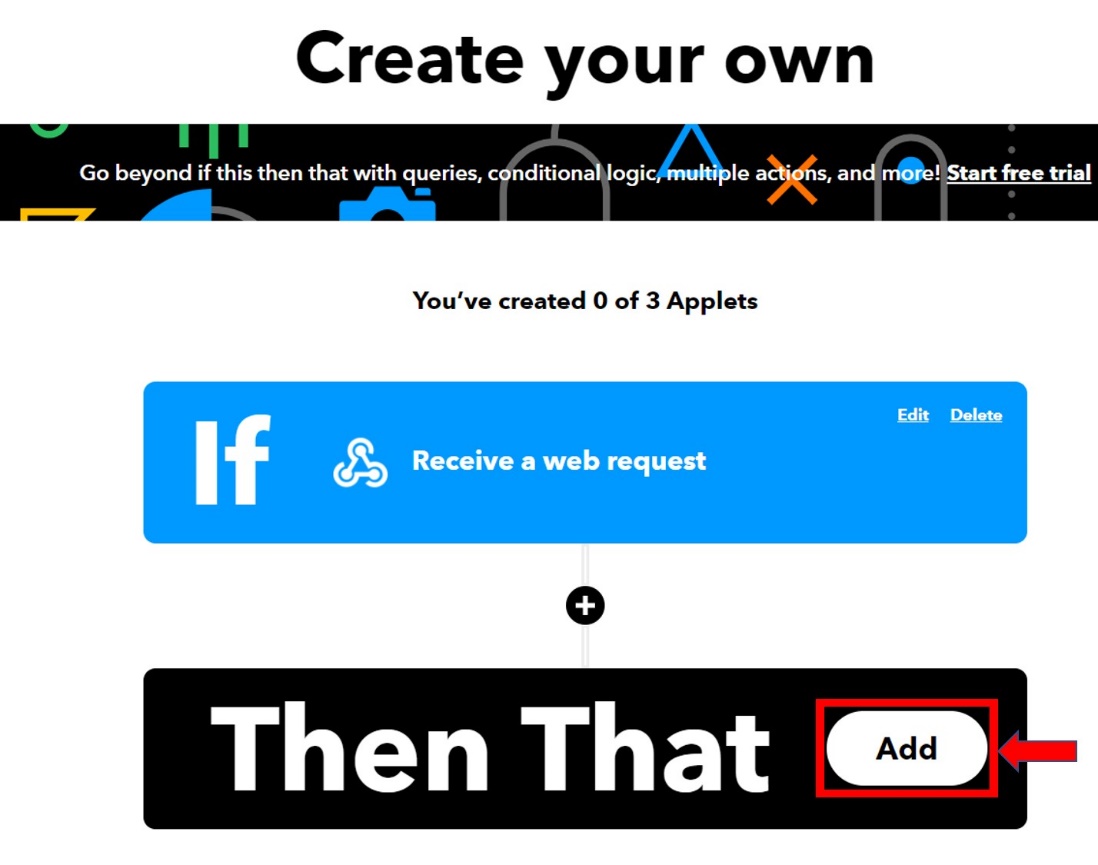
Next, choose the trigger as: ‘Receive a web request’ by clicking on it. Whenever webhooks will receive a web request, some action will take place. This we will define in the ‘THAT’ section.



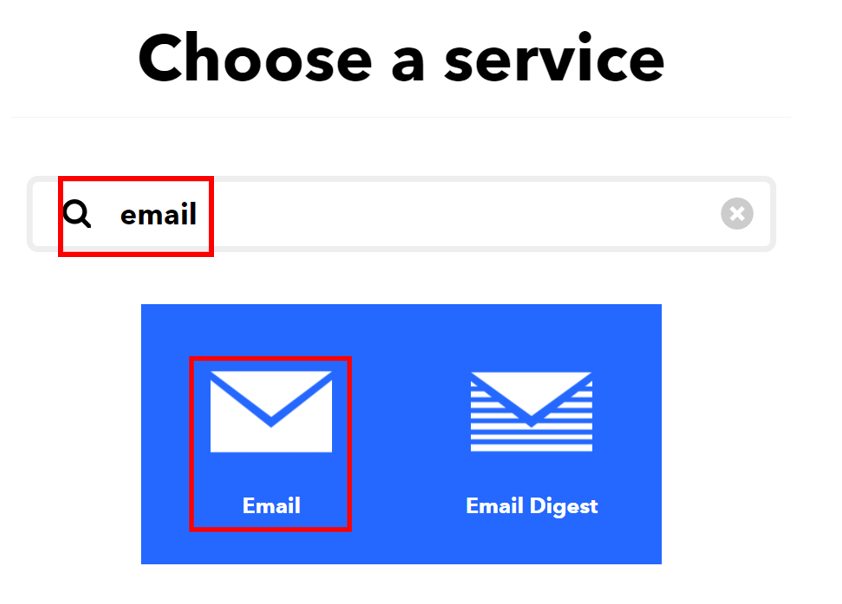
After clicking the Receive a web request, the following window will open up. We will write down **‘SEND EMAIL’** as the event name for the web request. You can use any other name of your choice. Click ‘Create Trigger’ button.



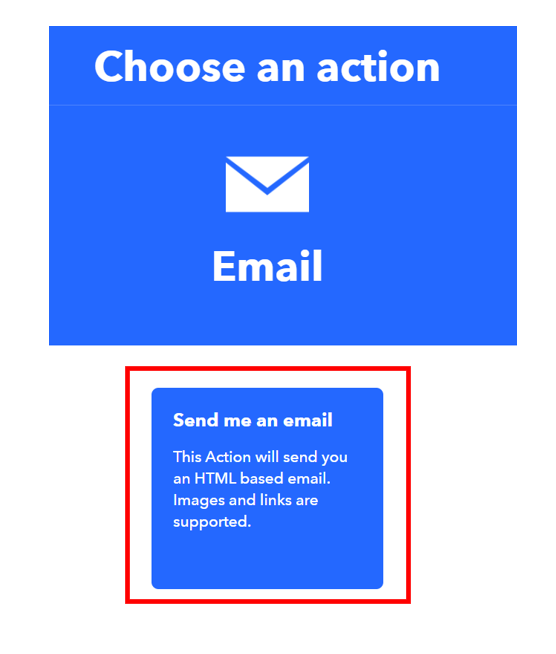
After the trigger is created, we are taken back to the web page where we first added the service for the ‘IF THIS’ section. Now we will click the ADD button for the ‘THEN THAT’ section.



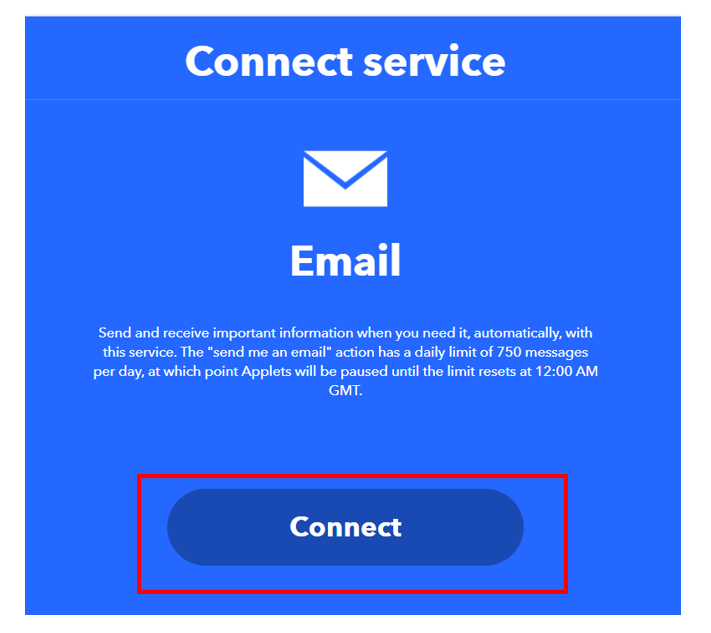
Now we will choose the service. We have to choose what will happen if a web request is received. We will type ‘email’ in the search option and click on its icon. This is because we want to receive email notification whenever a web request is received.



The following page opens up. Choose **‘Send me an email’** to proceed further.



Click on the ‘Connect’ button as shown below.

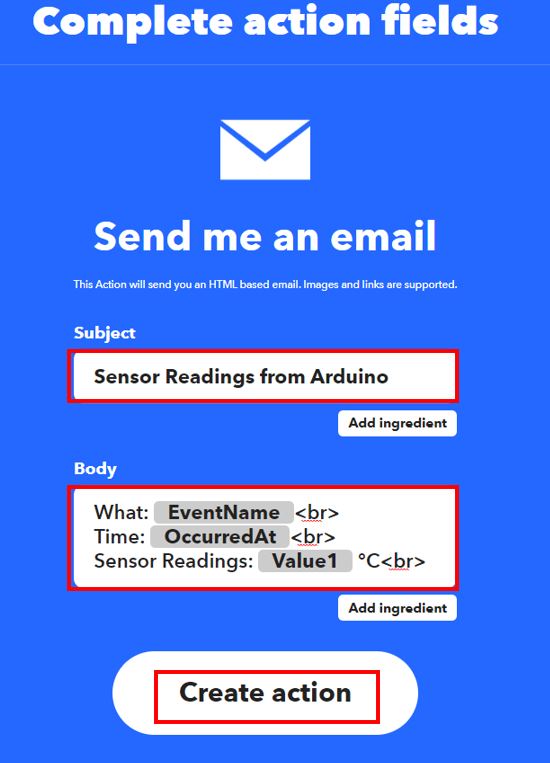


Next, write down your email address and click ‘Send Pin’ as shown below:



After you successfully enter the PIN, a new window will open up. Complete the action fields by specifying the subject and body of the email. Afterwards, click ‘Create Action.’

In our case, we want to send a temperature reading in the email so we have included the unit (°C) beside **Value1** in the body.

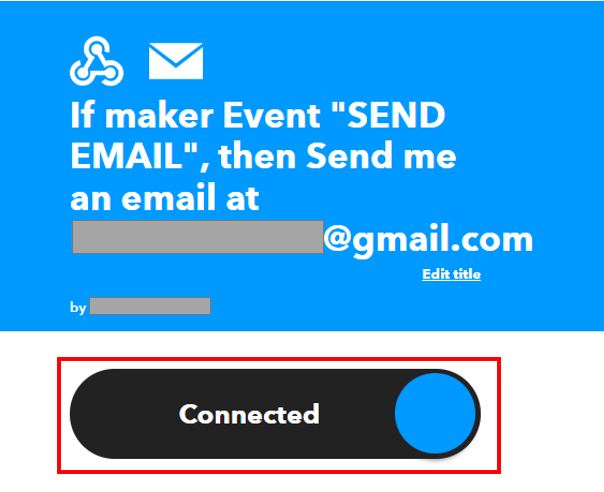


After we have created the action, we will be guided towards the initial web page of IFTTT. Click ‘Continue’ to proceed.



After this click the Finish button. Make sure to turn ON the notifications when the applet is running.

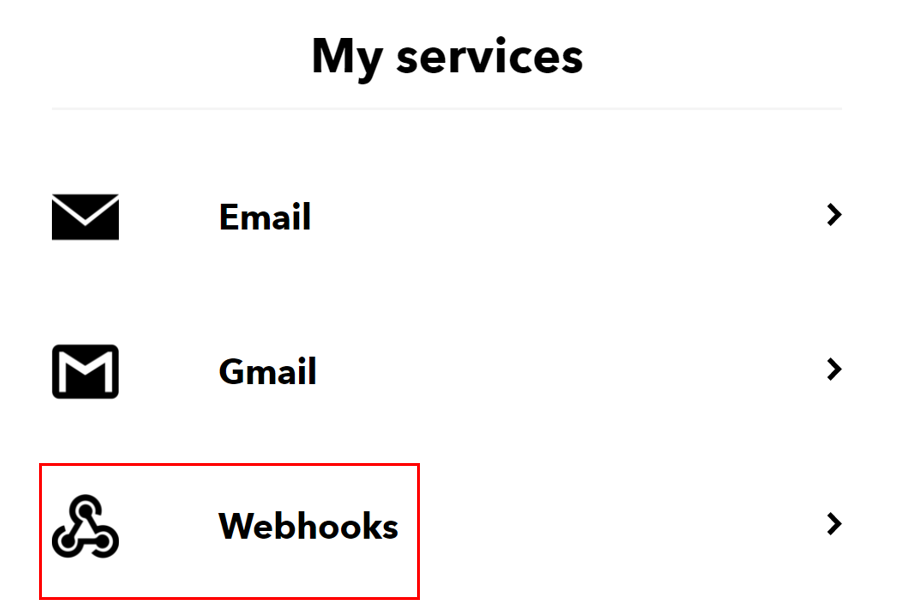
You have successfully created the applet as shown below.



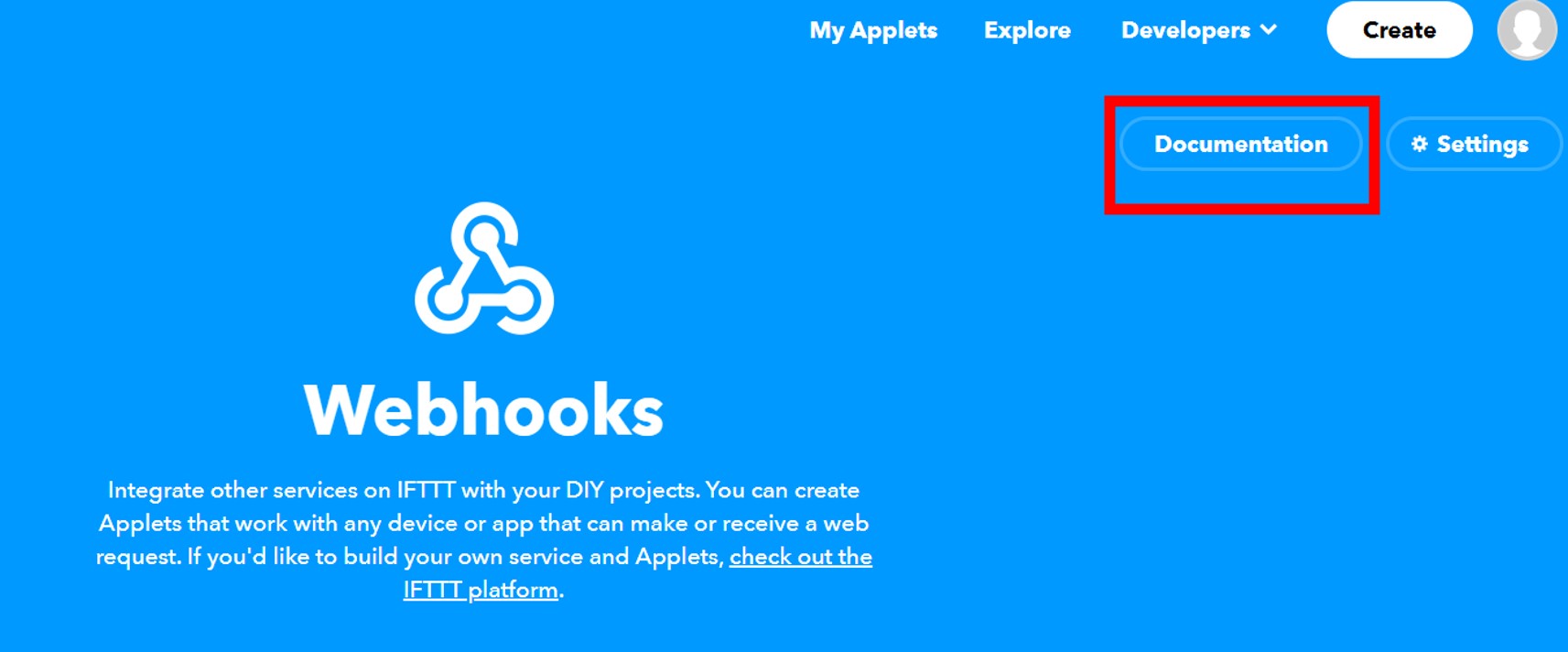
## Obtaining the Private Key

Before we proceed further with our project, we want to access our private key. This is important as it will be required while programming our Arduino board, in our case our TI MSP cards.

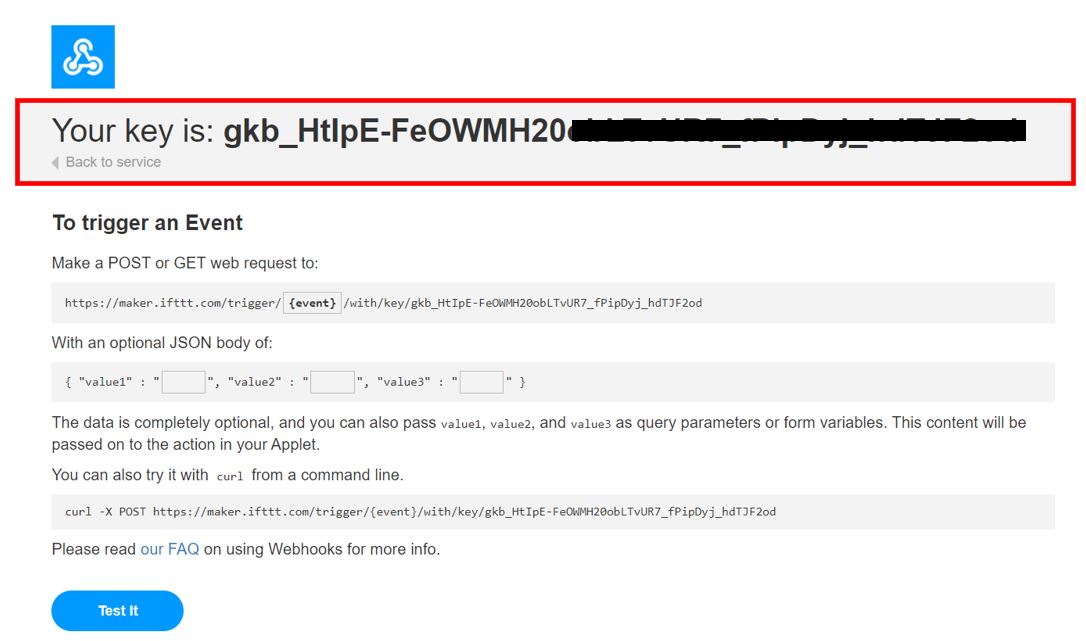
Go to your applet and select “My Services” or open a webpage with the link: **ifttt.com/my\_services**. The following windows will appear. Afterward, click on Webhooks.



This will take you to the following web page. Click on ‘Documentation.’



You will receive a key that should be secure with you.



## Testing the Applet

Before programming our Arduino, let us test the applet first. Open a new web browser and paste the following URL in the search bar and press enter:

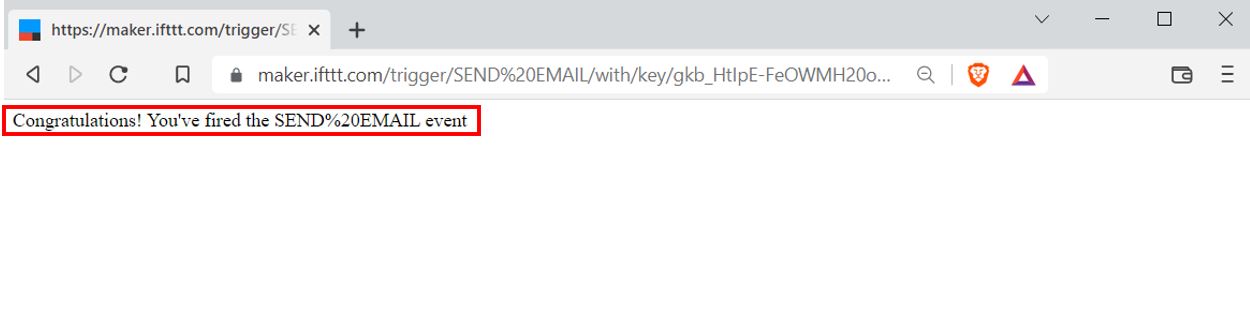
https://maker.ifttt.com/trigger/REPLACE\_WITH\_YOUR\_EVENT\_NAME/with/key/XXXXXXXXXXXXXXXXXXXXX?value1=31.2

Remember to change **REPLACE\_WITH\_YOUR\_EVENT\_NAME** with the event name that you set and the **XXXXXXXXXXXXXXXXXXXXX** with the Webhooks private key

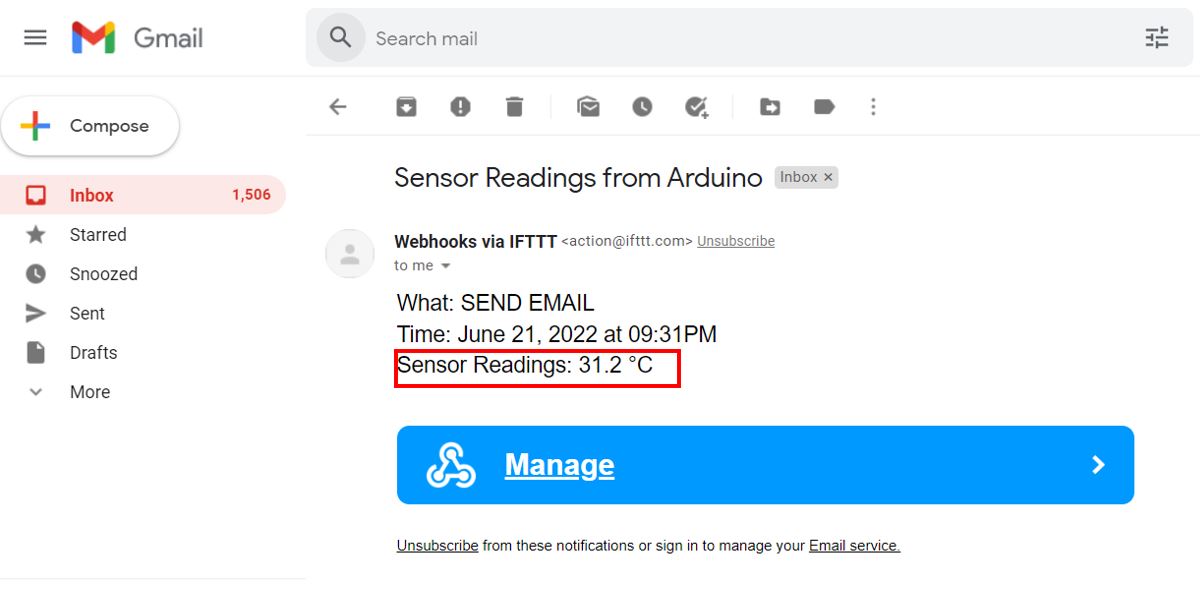
In our case, we are using the following URL:

https://maker.ifttt.com/trigger/SEND EMAIL/with/key/gkb\_HtIpE-FeOWMH20\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*?value1=31.2

After pressing enter, the web browser shows the following message:



Go to your email account and open it. There you will be able to view the email notification from IFTTT with the value you just sent .This means that the applet is running successfully.



### To trigger an Event with an arbitrary JSON payload

Make a POST or GET web request to:

https://maker.ifttt.com/trigger/**{event}**/**json**/with/key/xxxx-My-IFTTT-Unique-Key-xx*\* Note the extra****/json****path element in this trigger.*

With any JSON body. For example:

{ "this" : [ { "is": { "some": [ "test", "data" ] } } ] }

You can also try it with curl from a command line.

curl -X POST -H "Content-Type: application/json" -d '{"this":[{"is":{"some":["test","data"]}}]}' https://maker.ifttt.com/trigger/{event}/json/with/key/xxxx-My-IFTTT-Unique-Key-xx

Please read [our FAQ](http://ift.tt/webhooks_faq) on using Webhooks for more info.

Test It

### To trigger an Event with 3 JSON values

Make a POST or GET web request to:

https://maker.ifttt.com/trigger/**{event}**/with/key/xxxx-My-IFTTT-Unique-Key-xx

With an optional JSON body of:

{ "value1" : "", "value2" : "", "value3" : "" }

The data is completely optional, and you can also pass value1, value2, and value3 as query parameters or form variables. This content will be passed on to the action in your Applet.

You can also try it with curl from a command line.

curl -X POST https://maker.ifttt.com/trigger/{event}/with/key/xxxx-My-IFTTT-Unique-Key-xx

Please read [our FAQ](http://ift.tt/webhooks_faq) on using Webhooks for more info.

Test It

### To query a web service

You can query a publicly accessible HTTP endpoint using the Webhooks service.

The "Make a web request" query requires a URL and Method as query fields. The query optionally accepts a Content Type and Request Body as query fields.

The query will always provide the Status Code returned by the endpoint as an Ingredient.

In addition, if the endpoint returns JSON matching our expected format (shown below) we can parse the response and provide Value1, Value2, and Value3 ingredients from the response.

{ "value1" : "", "value2" : "", "value3" : "" }

## To Test

<https://maker.ifttt.com/trigger/SEND_EMAIL/with/key/xxxxMyKEYxxxxx?value1=31.2>

I get

Congratulations! You've fired the SEND\_EMAIL event

From CMD

**Curl -X POST**

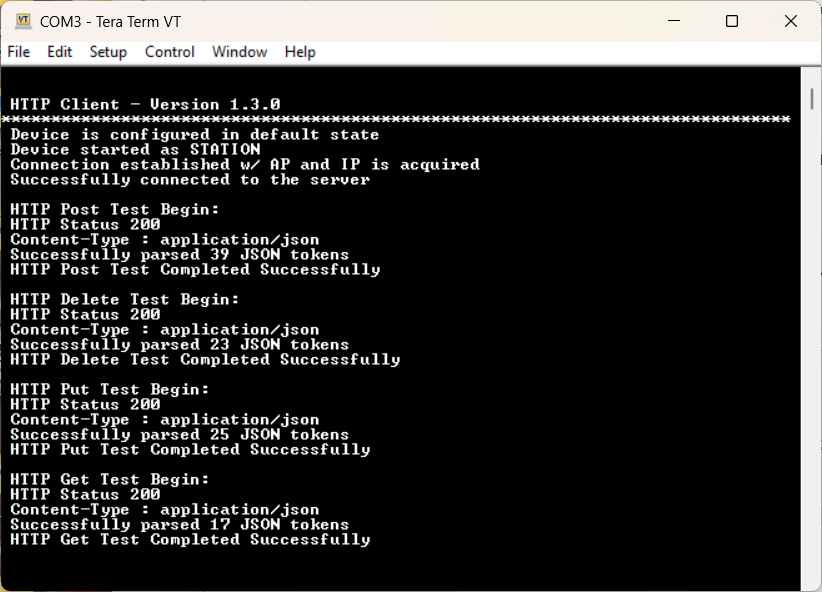
curl -X POST [http://maker.ifttt.com/trigger/SendEmailV2/with/key/ xxxxMyKEYxxxxx?value1=74.5](http://maker.ifttt.com/trigger/SendEmailV2/with/key/%20xxxxMyKEYxxxxx?value1=74.5)

WOW http works as get the email and data

# TI HTTP Client Example Test Out

TI http client example for F5529

Amazing works perfectly first time.



HTTP Client - Version 1.3.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Device is configured in default state

Device started as STATION

Connection established w/ AP and IP is acquired

Successfully connected to the server

**HTTP Post Test Begin:**

HTTP Status 200

Content-Type : application/json

Successfully parsed 39 JSON tokens

HTTP Post Test Completed Successfully

**HTTP Delete Test Begin:**

HTTP Status 200

Content-Type : application/json

Successfully parsed 23 JSON tokens

HTTP Delete Test Completed Successfully

**HTTP Put Test Begin:**

HTTP Status 200

Content-Type : application/json

Successfully parsed 25 JSON tokens

HTTP Put Test Completed Successfully

**HTTP Get Test Begin:**

HTTP Status 200

Content-Type : application/json

Successfully parsed 17 JSON tokens

HTTP Get Test Completed Successfully

## TI http\_client example with IFTTT PUT

**#define** POST\_DATA "{}" //Try with Post Data blank for "http://maker.ifttt.com"

**#define** DELETE\_REQUEST\_URI "/delete"

**#define** PUT\_REQUEST\_URI "/put"

**#define** PUT\_DATA "PUT request."

**#define** GET\_REQUEST\_URI "/get"

//#define HOST\_NAME "httpbin.org"

**#define** HOST\_NAME "maker.ifttt.com"

**#define** PUT\_REQUEST\_URI "/trigger/SendEmailV2/with/key/xxxxxxMyIFTTTKeyxxxxxx?value1=GenAmmeter&value2=AmpGnPoHv"

HTTP Client - Version 1.3.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Device is configured in default state

Device started as STATION

Connection established w/ AP and IP is acquired

Successfully connected to the server

HTTP Put Test Begin:

HTTP Status 200

Content-Type : application/json

**HTTP Put Test Completed Successfully**

**From:** Webhooks via IFTTT <action@ifttt.com>   
**Sent:** Saturday, April 8, 2023 4:40 PM  
**To:** xxxxMyEmailAddressxxxx  
**Subject:** The event named "GenAmmeter" occurred on the Maker Webhooks service SendEmailV2

What: SendEmailV2  
When: April 8, 2023 at 04:39PM  
Extra Data: AmpGnPoHv

# Merging TI HTTP example back into My Application Attic Fan program with Get Time and Email WiFi code

## emailAndGetTime.c: Merged GetTime, Send Email, and IFTTT Send Email code

Other files in this Products application are measuring 6 different temperatures using the TI LMT01 temperature sensors, and other sensors to provide optimized powered Attic Fan On/Off control and HVAC control. At application defined times in myTimers.c ; Summary Temperature, On/OFF status, Times, etc is gathered into two 64 byte strings and it then sets a flag to “Get Timer Server Time” or send a Status Email via IFTTT. Main.c sees these flags and makes a function call to “emailAndGetTime.c”. The benefit of this is myTimers and Applications keeps running while main.c makes the function calls to Get the Time or Send a Status Email.

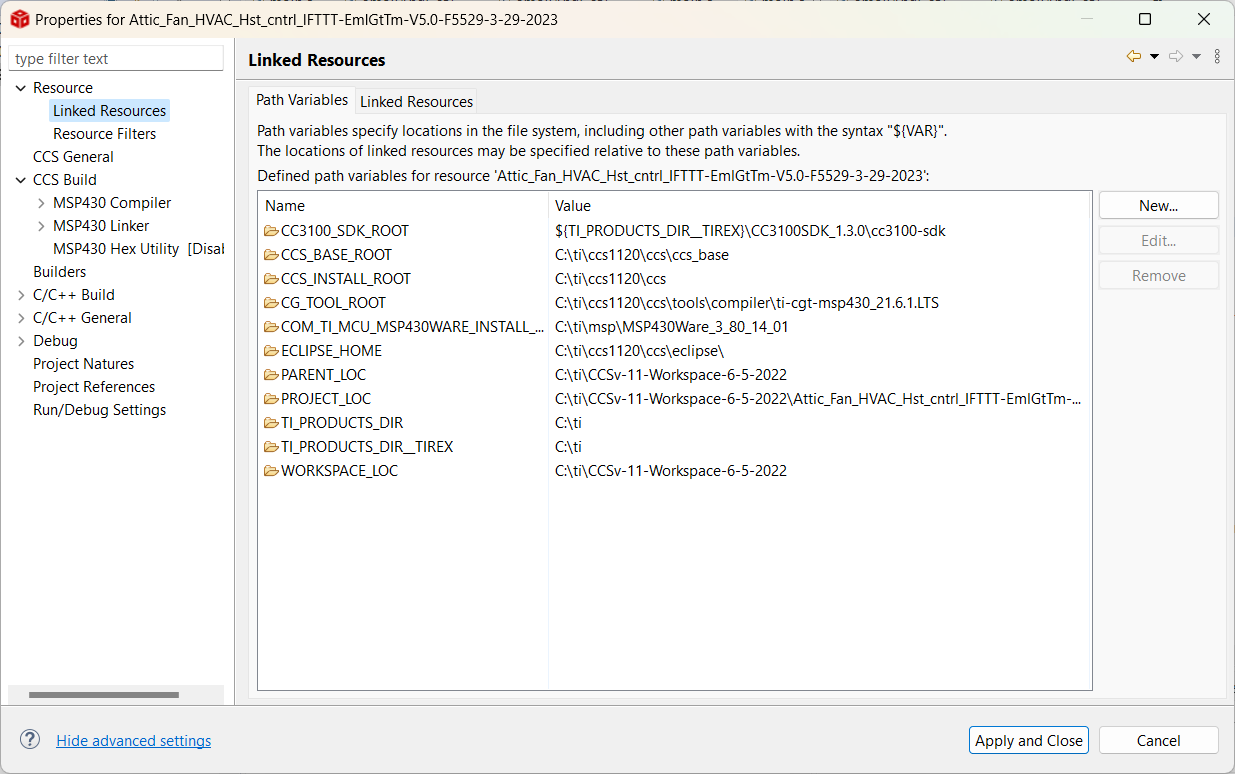
I created a new .c and .h file called **“emailAndGetTime.c”** and **“emailAndGetTime.h”.** This is 95% TI example code with my function calls from main. The Time data is returned to set the RT(Real Time Clock), etc in my application. The IFTTT PUT uses one very long URI with my unique IFTTT userID, the EventName, and the Application Summary Data. (See Attached files)

The file jsm.c needs to be copied from the http\_client example and pasted into your project.

## Properties

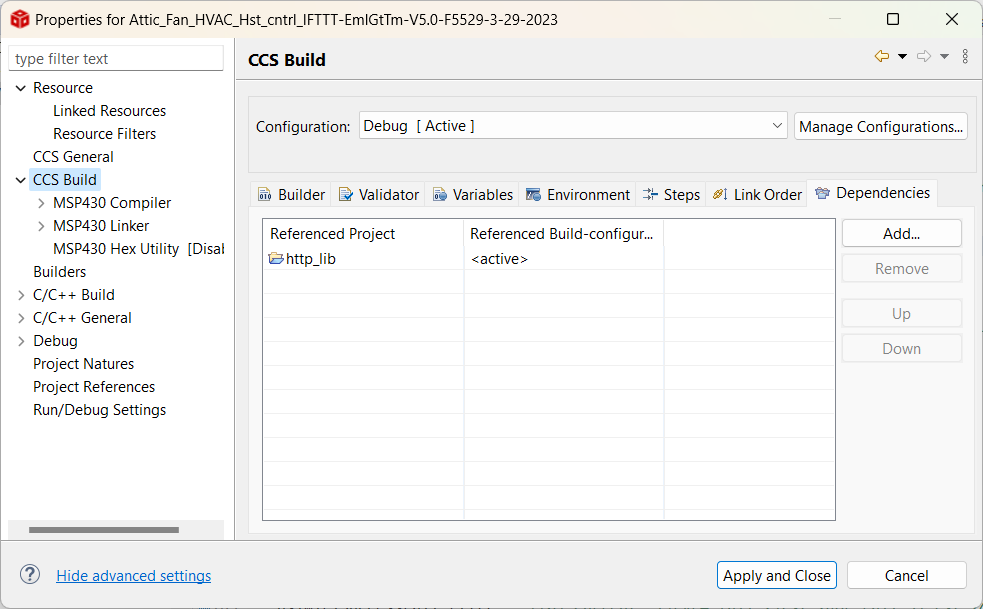
A lot of careful setup needs to be done by Right Clicking on your project and going to **Properties,** to update the files / setting to the items that were used in http\_client example.

### Resource Linked Resources

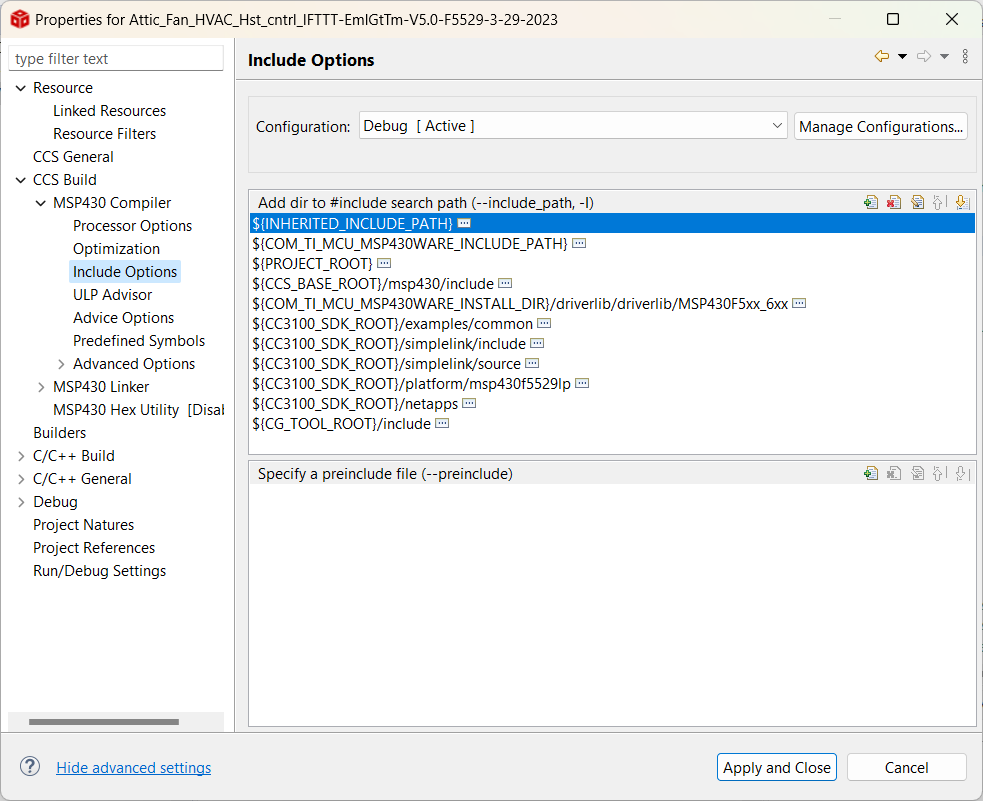


### CCS Build Dependencies

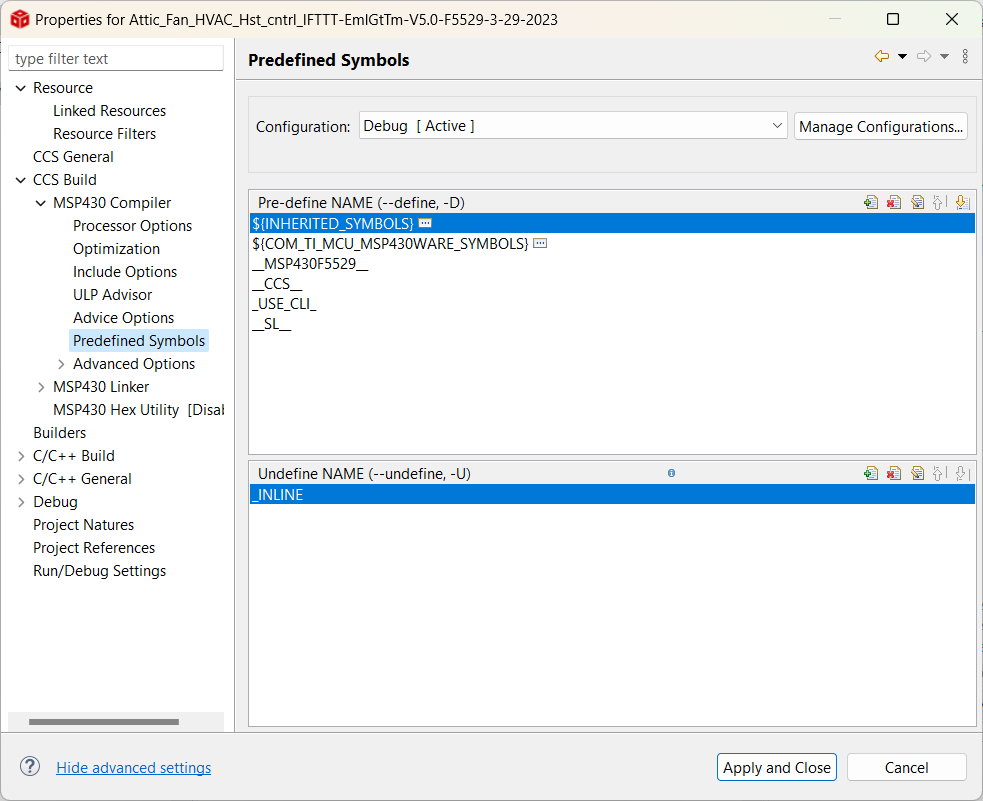
Under CCS Build, Dependencies Add http\_lib and make it active



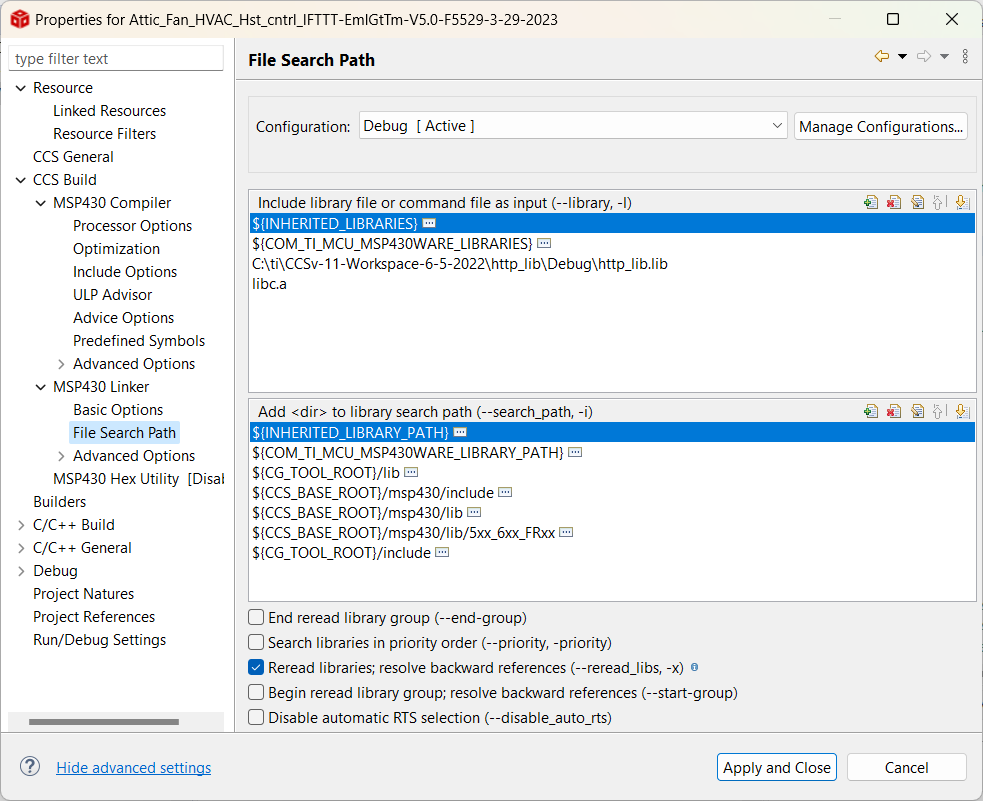
### CCS Build MSP430 Compiler Include Options



### CCS Build MSP430 Compiler Predefined Symbols



### Linker File Search Path



## Config.h merges

/\* Application specific status/error codes \*/

**typedef** **enum**{

*DEVICE\_NOT\_IN\_STATION\_MODE* = -0x7D0, /\* Choosing this number to avoid overlap w/ host-driver's error codes \*/

*EMAIL\_SET\_INVALID\_MESSAGE* = *DEVICE\_NOT\_IN\_STATION\_MODE* - 1,

*EMAIL\_SET\_INVALID\_CASE* =*EMAIL\_SET\_INVALID\_MESSAGE* - 1,

*EMAIL\_CONNECT\_INVALID\_CONFIURATION* = *EMAIL\_SET\_INVALID\_CASE* - 1,

*TCP\_RECV\_ERROR* = *EMAIL\_CONNECT\_INVALID\_CONFIURATION* - 1,

*TCP\_SEND\_ERROR* = *TCP\_RECV\_ERROR* - 1,

*SMTP\_ERROR* = *TCP\_SEND\_ERROR*,

*SMTP\_INVALID\_CASE* = *SMTP\_ERROR* -1,

// Added from get timeLH 10-28-2020

*SNTP\_SEND\_ERROR* = *DEVICE\_NOT\_IN\_STATION\_MODE* - 1,

*SNTP\_RECV\_ERROR* = *SNTP\_SEND\_ERROR* - 1,

*SNTP\_SERVER\_RESPONSE\_ERROR* = *SNTP\_RECV\_ERROR* - 1,

// Added from emailAndGetTime.c http\_client lead typedef

*INVALID\_HEX\_STRING* = *DEVICE\_NOT\_IN\_STATION\_MODE* - 1,

*FILE\_NOT\_FOUND\_ERROR* = *TCP\_SEND\_ERROR* - 1,

*INVALID\_SERVER\_RESPONSE* = *FILE\_NOT\_FOUND\_ERROR* - 1,

*FORMAT\_NOT\_SUPPORTED* = *INVALID\_SERVER\_RESPONSE* - 1,

*FILE\_WRITE\_ERROR* = *FORMAT\_NOT\_SUPPORTED* - 1,

*INVALID\_FILE* = *FILE\_WRITE\_ERROR* - 1,

*STATUS\_CODE\_MAX* = -0xBB8

}e\_AppStatusCodes;

## http\_lib, httpcli.c Changes

/\* Configurable lengths \*/

**#define** CONTENT\_BUFLEN 128

//#define CONTENT\_BUFLEN 256

//#define URI\_BUFLEN 128 //This is TI default setting

**#define** URI\_BUFLEN 256 //4-8-2023 This will allow a longer URI for IFTTT post

//#define SEND\_BUFLEN 128

**#define** SEND\_BUFLEN 256

**#define** MAX\_FIELD\_NAME\_LEN 24

**#define** STATUS\_BUFLEN 16

## Function call to send an email via IFTTT

**int** **LHIFTTTSendEamil**() // 4-18-2023

{

usiEmailDummy = 0;

usiEmail1xTime2xIFTTTemail3 = 3; //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

//Load Product Name String into 214 character URI

usiURIarrayCt = 60; //value= = is character 61

**for** (usiDataarrayCt = 0; usiDataarrayCt <=14; usiDataarrayCt++){ //So 15 times loop as don't want to load the null

PUT\_REQUEST\_URI[usiURIarrayCt] = LHProd15CharNam [usiDataarrayCt];

usiURIarrayCt++;

}

//Load Subject Data String into 214 character URI

usiURIarrayCt = 83; //value= = is character 82

**for** (usiDataarrayCt = 0; usiDataarrayCt <=62; usiDataarrayCt++){ //So 63 times loop as don't want to load the null

PUT\_REQUEST\_URI[usiURIarrayCt] = LHemailSubject [usiDataarrayCt];

usiURIarrayCt++;

}

//Load Body Data String into 214 character URI

PUT\_REQUEST\_URI[usiURIarrayCt] = '-'; //Space Between Subject and Body

usiURIarrayCt++;

**for** (usiDataarrayCt = 0; usiDataarrayCt <=62; usiDataarrayCt++){ //So 63 times loop as don't want to load the null

PUT\_REQUEST\_URI[usiURIarrayCt] = LHemailBody [usiDataarrayCt];

usiURIarrayCt++;

}

//PUT\_REQUEST\_URI

\_i32 retVal = -1;

HTTPCli\_Struct httpClient;

retVal = initializeAppVariables();

ASSERT\_ON\_ERROR(retVal);

/\* Configure command line interface \*/

CLI\_Configure();

displayBanner();

/\*

\* Following function configures the device to default state by cleaning

\* the persistent settings stored in NVMEM (viz. connection profiles &

\* policies, power policy etc)

\*

\* Applications may choose to skip this step if the developer is sure

\* that the device is in its default state at start of application

\*

\* Note that all profiles and persistent settings that were done on the

\* device will be lost

\*/

retVal = configureSimpleLinkToDefaultState();

**if**(retVal < 0)

{

**if** (*DEVICE\_NOT\_IN\_STATION\_MODE* == retVal)

{

CLI\_Write(" Failed to configure the device in its default state \n\r");

}

LOOP\_FOREVER();

}

CLI\_Write(" Device is configured in default state \n\r");

/\*

\* Initializing the CC3100 device

\* Assumption is that the device is configured in station mode already

\* and it is in its default state

\*/

retVal = sl\_Start(0, 0, 0);

**if** ((retVal < 0) ||

(*ROLE\_STA* != retVal) )

{

CLI\_Write(" Failed to start the device \n\r");

LOOP\_FOREVER();

}

CLI\_Write(" Device started as STATION \n\r");

/\* Connecting to WLAN AP \*/

retVal = establishConnectionWithAP();

**if**(retVal < 0)

{

CLI\_Write(" Failed to establish connection w/ an AP \n\r");

LOOP\_FOREVER();

}

CLI\_Write(" Connection established w/ AP and IP is acquired \n\r");

/\* Connect to HTTP server \*/

retVal = ConnectToHTTPServer(&httpClient);

**if**(retVal < 0)

{

LOOP\_FOREVER();

}

CLI\_Write("\n\r");

CLI\_Write("\n\r");

CLI\_Write(" HTTP Put Test Begin:\n\r");

retVal = HTTPPutMethod(&httpClient);

**if**(retVal < 0)

{

CLI\_Write(" HTTP Put Test failed.\n\r");

}

CLI\_Write(" HTTP Put Test Completed Successfully\n\r");

CLI\_Write("\n\r");

/\* Stop the CC3100 device \*/

retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

**if**(retVal < 0)

{

LOOP\_FOREVER();

}

**return** SUCCESS;

} // Closing bracket for int LHIFTTTSendEamil()

### Tera Term Log

HTTP Client - Version 1.3.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Device is configured in default state

Device started as STATION

Connection established w/ AP and IP is acquired

Successfully connected to the server

HTTP Put Test Begin:

HTTP Status 200

Content-Type : application/json

HTTP Put Test Completed Successfully

### Webhooks IFTTT Email received

**Can’t use spaces in the Data stream as IFTTT treats them as nulls, so will not work**

Product Name now added in as Value 1 during run

**From:** Webhooks via IFTTT <action@ifttt.com>   
**Sent:** Tuesday, April 25, 2023 2:26 PM  
**To:** myemailaddress@xxx.com  
**Subject:** The event named "LwGrAtticFnHVAC" occurred on the Maker Webhooks service SendEmailV2

What: SendEmailV2  
When: April 25, 2023 at 02:26PM  
Extra Data: LAtFnHVA-m01d01-t0012-S750N000-CD000-F000-P00-G00m00-G07575c00x-A07575-E07575-I7575-R07575-F0t000L0012-H07575c00hy000t000e35-00-ZX

## 1st Working IFTTT on F5529 GetTime and SendEmail via IFTTT

Get time application - Version 1.3.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Device is configured in default state

Device started as STATION

Connection established w/ AP and IP is acquired

LH Just after get HostIP

LH Just after createConnection

LH Just BEFORE getSNTPTTime...

Server time-a.nist.gov has responded with time information

Tue Apr 25 2023 14:39:6

LH Just AFTER getSNTPTTime...

Device disconnected from the AP on application's request

HTTP Client - Version 1.3.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Device is configured in default state

Device started as STATION

Connection established w/ AP and IP is acquired

Successfully connected to the server

HTTP Put Test Begin:

HTTP Status 200

Content-Type : application/json

HTTP Put Test Completed Successfully

# Full emailAndGetTime code

## Data for Email into Two 64 byte Strings

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function Email SubjectBodyAssemble

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** SubjectBodyAssemble (**void**)

{

//64 characters max, included end Null Character

//Since count from zero, 64 characters is LHemailSubject[63] = '\0'; or LHemailBody[63]

//Note the benefit of using the function IntegerToCharacters () is that if number is greater than say 9 then only LSD written

//No over flow if eg number = 12, IntegerToCharacters () will give 2 for is digit, 12+ 48 = ASCII 60 = some special character

//4-23-2023 for IFTTT string cannot have spaces as it treats it as a null, so change to -

**if** (usiLwrOrUprAticFan == 0){

LHemailSubject[0] = 'L'; //Lower Attic Fan /HVAC

}

**else** **if** (usiLwrOrUprAticFan == 1){

LHemailSubject[0] = 'U'; //Upper Attic Fan / HVAC

}

**else** **if** (usiLwrOrUprAticFan == 2){

LHemailSubject[0] = 'C'; //Cardboard test unit

}

LHemailSubject[1] = 'A';

LHemailSubject[2] = 't';

LHemailSubject[3] = 'F';

LHemailSubject[4] = 'n';

LHemailSubject[5] = 'H';

LHemailSubject[6] = 'V';

LHemailSubject[7] = 'A';

LHemailSubject[8] = '-';

LHemailSubject[9] = 'M';

IntegerToCharacters (usiMonthOfYearGetTime);

LHemailSubject[10] = Char2;

LHemailSubject[11] = Char1;

LHemailSubject[12] = 'D';

// i = usiMth0Day % 10; //Give ones digit of a big number

// LHemailSubject[14] = i + 48; //Tens digit of day

// numRemain = usiMth0Day / 10;

// i = numRemain % 10; //Give ones digit of a big number

// LHemailSubject[13] = i + 48; //Ones digit of day

// LHemailSubject[15] = '-';

IntegerToCharacters (usiDayOfMonthGetTime);

LHemailSubject[13] = Char2;

LHemailSubject[14] = Char1;

LHemailSubject[15] = '-';

// 0= Intial value, 1 = NIST Server Time, 2 = Photo Cell Time; set to 2 if NIST not available

**if** (usiNISTorPhtCelTime == 1){

LHemailSubject[16] = 'T'; //Upper Case T for NIST based time

IntegerToCharacters (usiHrsAfterMidnightMyZone);

LHemailSubject[17] = Char2; //Hours 10's digit

LHemailSubject[18] = Char1; //Hours 1's digit

IntegerToCharacters (usiMinRmdrAfterMidnightMyZone);

LHemailSubject[19] = Char2; //Minutes 10's digit

LHemailSubject[20] = Char1; //Minutes 1's digit

}

**else** **if** (usiNISTorPhtCelTime == 2){

LHemailSubject[9] = 'm'; //Over write above to Lower Case as Photo Cell based

LHemailSubject[12] = 'd'; //Over write above to Lower Case as Photo Cell based

LHemailSubject[16] = 't'; //Lower Case T for Photo cell based time

IntegerToCharacters (usiHrsAfterMidnightPhCel);

LHemailSubject[17] = Char2; //Hours 10's digit

LHemailSubject[18] = Char1; //Hours 1's digit

IntegerToCharacters (usiMinRmdrAfterMidnightPhCel);

LHemailSubject[19] = Char2; //Minutes 10's digit

LHemailSubject[20] = Char1; //Minutes 1's digit

}

LHemailSubject[21] = '-';

LHemailSubject[22] = 'S'; //Sunlight minutes

IntegerToCharacters (usiPrevDaylightSec / 60); //Previous days sunlight minutes, used to adjust time

LHemailSubject[23] = Char3;

LHemailSubject[24] = Char2;

LHemailSubject[25] = Char1;

**if** ((usiHrsAfterMidnightPhCel >= 8) && (usiHrsAfterMidnightPhCel <= 18)) {

LHemailSubject[26] = 'R'; //Sun Rise Time

IntegerToCharacters (usiRiseHr);

LHemailSubject[27] = Char1; //Hours 1's digit

IntegerToCharacters (usiRiseMin);

LHemailSubject[28] = Char2; //Minutes 10's digit

LHemailSubject[29] = Char1; //Minutes 1's digit

LHemailSubject[30] = '-';

}

**else** {

LHemailSubject[26] = 'N'; //Sun Set Time

**if** (usiSetHr >= 13){ //Set Hour should always be greater than 13, like 5 PM in winter is 1700, but guard against negative number

IntegerToCharacters (usiSetHr - 12);

}

**else** **if** (usiSetHr <= 12){

IntegerToCharacters (usiSetHr);

}

LHemailSubject[27] = Char1; //Hours 1's digit

IntegerToCharacters (usiSetMin);

LHemailSubject[28] = Char2; //Minutes 10's digit

LHemailSubject[29] = Char1; //Minutes 1's digit

LHemailSubject[30] = '-';

}

LHemailSubject[31] = 'C'; //Continouous, not reset monthly, reset on POR, power outage, reset when re-program micro,

LHemailSubject[32] = 'D'; //Continouous,, not reset monthly, reset when re-program micro

i = (**unsigned** **short**)(usiSecContLong / 86400); //65,535 for short sent / 24Hr/day = 2,730 days!! if loged hours, but loging days

IntegerToCharacters (i);

LHemailSubject[33] = Char3;

LHemailSubject[34] = Char2;

LHemailSubject[35] = Char1;

LHemailSubject[36] = '-';

LHemailSubject[37] = 'F';

i = (**unsigned** **short**)usiFlashWriteCntEvr; //# times ever wrote to Flash

IntegerToCharacters (i);

LHemailSubject[38] = Char3;

LHemailSubject[39] = Char2;

LHemailSubject[40] = Char1;

LHemailSubject[41] = '-';

LHemailSubject[42] = 'P'; //POR count in last month

IntegerToCharacters (usiPORcount); //POR reset counts; back to zero at end of each month

LHemailSubject[43] = Char2;

LHemailSubject[44] = Char1;

LHemailSubject[45] = '-';

LHemailSubject[46] = 'G'; //Get Time Hours

IntegerToCharacters (usiHrsGetTime);

LHemailSubject[47] = Char2;

LHemailSubject[48] = Char1;

LHemailSubject[49] = 'm'; //Get time minutes

IntegerToCharacters (usiMinutesGetTime);

LHemailSubject[50] = Char2;

LHemailSubject[51] = Char1;

LHemailSubject[52] = '-';

**if** (usiLwrOrUprAticFan == 0){

LHemailSubject[53] = 'G'; //HVAC2, Garage for the Lower Attic Fan controller

}

**else** **if** (usiLwrOrUprAticFan == 1){

LHemailSubject[53] = 'H'; //HVAC2, Garage for the Lower Attic Fan controller

}

**else** **if** (usiLwrOrUprAticFan == 2){

LHemailSubject[53] = 'C'; //Cardboard test unit

}

i = (usiTempCntAvgMax24[5]\*36/32)-585; //HVAC2 Duct (5), 24Hr Max, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //HVAC2 duct (5) last 24 Hr Max

LHemailSubject[54] = Char3;

LHemailSubject[55] = Char2;

LHemailSubject[56] = Char1;

i = (usiTempCntAvgMin24[5]\*36/32)-585; //HAVC2 duct (5), 24Hr Min, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //HVAC2 duct (5) last 24 Hr Minutes

LHemailSubject[57] = Char2;

LHemailSubject[58] = Char1;

LHemailSubject[59] = 'c'; //HVAC1 On count

//usiHVAC2Cycles [6]; //21 usiHVAC2Cycles 0)Evr AC 1)24Hr AC 2)24Hr Heat 3)Evr Heat 4)0 to 10 cycles counter 5)Prev Day AC cycles

// IntegerToCharacters (usiHVAC2Cycles [1] + usiHVAC2Cycles [2]); //These are inflated due to Thermostata On/Off during Hyst Hold Off

IntegerToCharacters (usiHVAC2TruOnNotHysHldOff24HrCt);

LHemailSubject[60] = Char2;

LHemailSubject[61] = Char1;

LHemailSubject[62] = 'x';

LHemailSubject[63] = '\0'; //last character should already be null, count from 0 so 63 is char 64

printf("\nMessage Subject is: %s\n", LHemailSubject);

usiDummy = 0;

// ConvertToChar = usiTest3 + 48;

// LHemailBody[9] = ConvertToChar;

// Attic Tempearture (0)

LHemailBody[0] = 'A'; //Attic Temp Max and Min last 24 hours

// i = (usiTempCntAvgMax24[0]\*36/32)-585; //Attic Sensor 0, 24Hr Max, 3 digit number, 1's digit is tenths

//Any number >1820 = short overflow, 1820 x 36 = 65520, 1820 = 146.2 Deg F

//So need to use a Long to do the Calculation; but should only need it for the Roof

usiConvertLong0 = usiTempCntAvgMax24[0];

usiConvertLong0 = (usiConvertLong0 \*36/32)-585; //Roof Sensor3, 24Hr Max, 3 digit number, 1's digit is ones

i = (**short**)usiConvertLong0;

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //Attic last 24 Hr Max

LHemailBody[1] = Char3; //3 Digit as often over 100 DegF

LHemailBody[2] = Char2;

LHemailBody[3] = Char1;

i = (usiTempCntAvgMin24[0]\*36/32)-585; //Attic Sensor 0, 24Hr Min, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //Upper Cabinet last 24 Hr Max

LHemailBody[4] = Char2;

LHemailBody[5] = Char1;

LHemailBody[6] = '-';

// Exterior Temperature (1)

LHemailBody[7] = 'E'; //Exterior Temp Max and Min last 24 hours

i = (usiTempCntAvgMax24[1]\*36/32)-585; //Exterior Sensor 1(b1), 24Hr Max, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //Exteriort last 24 Hr Max

LHemailBody[8] = Char3;

LHemailBody[9] = Char2;

LHemailBody[10] = Char1;

i = (usiTempCntAvgMin24[1]\*36/32)-585; //Exterior Sensor 1, 24Hr Min, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //Exterior last 24 Hr Max

LHemailBody[11] = Char2;

LHemailBody[12] = Char1;

LHemailBody[13] = '-';

// Interior Temperature

LHemailBody[14] = 'I'; //Interior Room Temp Max and Min last 24 hours

i = (usiTempCntAvgMax24[2]\*36/32)-585; //Intterior Room Sensor 2, 24Hr Max, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //Interior last 24 Hr Max

LHemailBody[15] = Char2;

LHemailBody[16] = Char1;

i = (usiTempCntAvgMin24[2]\*36/32)-585; //Interior 2, 24Hr Min, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //Interior 2 last 24 Hr Max

LHemailBody[17] = Char2;

LHemailBody[18] = Char1;

LHemailBody[19] = '-';

// Roof Temperature (3)

LHemailBody[20] = 'R'; //Roof Temp Max and Min last 24 hours

//Any number >1820 = short overflow, 1820 x 36 = 65520, 1820 = 146.2 Deg F

//So need to use a Long to do the Calculation; but should only need it for the Roof

usiConvertLong0 = usiTempCntAvgMax24[3];

usiConvertLong0 = (usiConvertLong0 \*36/32)-585; //Roof Sensor3, 24Hr Max, 3 digit number, 1's digit is ones

i = (**short**)usiConvertLong0;

**if** (i % 10 >= 5) { //% 10 gives the 1's digit, really the 0.xDegF digit, This is to round up or round down

i = (i /10) + 1; //Divide by 10 as only send 1's and 10's digit, add 1 if decimal >=5

}

**else**{

i = i / 10; //Divide by 10 as only send 1's and 10's digit, add 1 if decimal >=5

}

IntegerToCharacters (i); //Roof last 24 Hr Max

LHemailBody[21] = Char3;

LHemailBody[22] = Char2;

LHemailBody[23] = Char1;

i = (usiTempCntAvgMin24[3]\*36/32)-585; //Roof Sensor 3, 24Hr Min, 3 digit number, 1's digit is ones

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //Exterior last 24 Hr Max

LHemailBody[24] = Char2;

LHemailBody[25] = Char1;

LHemailBody[26] = '-';

// Fans

LHemailBody[27] = 'F'; //Fans

LHemailBody[28] = usiFan24HrOnCt + 48; //Fan Off to On Count last 24 hours, reset every midnight

LHemailBody[29] = 't'; //Fan On Hour xx.x

IntegerToCharacters (usiFanOn24Min / 6); //Fan On Hours xx.x last 24 Hr, divide by 6 not 60 so create pseudo decimal

LHemailBody[30] = Char3;

LHemailBody[31] = Char2;

LHemailBody[32] = Char1;

// volatile unsigned long usiFanLastOnSecAftMidnight = 0;

// volatile unsigned long usiFanLastOffSecAftMidnight = 0;

// Can leave as Photo Cell Hours, as even if NIST time, Photo Cell time is updated from NIST time

**if** ((usiHrsAfterMidnightPhCel >= 8) && (usiHrsAfterMidnightPhCel <= 18)) { //1 PM email

LHemailBody[33] = 'S'; //Last Fan start time

IntegerToCharacters (usiFanLastOnHr0Min);

LHemailBody[34] = Char5;

LHemailBody[35] = Char4; //Note Char 3 is the zero in Hr0Min format

LHemailBody[36] = Char2;

LHemailBody[37] = Char1;

}

**else**{ //Midnight email will send off hour, but if Fans off after midnight will not get the time, but will see total time On

//Ah but if not Off then will get the last time Off was recorded, could be days before

LHemailBody[33] = 'L'; //Last Fan off time

IntegerToCharacters (usiFanLastOffHr0Min);

LHemailBody[34] = Char5;

LHemailBody[35] = Char4; //Note Char 3 is the zero in Hr0Min format

LHemailBody[36] = Char2;

LHemailBody[37] = Char1;

}

LHemailBody[38] = '-';

LHemailBody[39] = 'H'; //HVAC1

i = (usiTempCntAvgMax24[4]\*36/32)-585; //HVAC1 Duct (4), 24Hr Max, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //HVAC1 duct (4) last 24 Hr Max

LHemailBody[40] = Char3;

LHemailBody[41] = Char2;

LHemailBody[42] = Char1;

i = (usiTempCntAvgMin24[4]\*36/32)-585; //HAVC1 duct (4), 24Hr Min, 3 digit number, 1's digit is tenths

**if** (i % 10 >= 5) {

i = (i /10) + 1;

}

**else**{

i = i / 10;

}

IntegerToCharacters (i); //HVAC1 duct (4) last 24 Hr Minimum

LHemailBody[43] = Char2;

LHemailBody[44] = Char1;

LHemailBody[45] = 'c'; //HVAC1 On count

//usiHVAC1Cycles [6]; //21 usiHVAC1Cycles 0)Evr AC 1)24Hr AC 2)24Hr Heat 3)Evr Heat 4)0 to 10 cycles counter 5)Prev Day AC cycles

// IntegerToCharacters (usiHVAC1Cycles [1] + usiHVAC1Cycles [2]);

IntegerToCharacters (usiHVAC1TruOnNotHysHldOff24HrCt);

LHemailBody[46] = Char2;

LHemailBody[47] = Char1;

LHemailBody[48] = 'h'; //HVAC1 hysteresi hold off xx.x hours

LHemailBody[49] = 'y'; //HVAC1 hysteresi hold off xx.x hours

//usiHysHoldOffTime[5]; [0]Lst Hys H Off Sec(Log Min), [1]Lst 10 Cycle Avg Sec(Log Min), [2]Lst 24Hr Avg Sec (Log Min),

//[3]Lst24Hr Hys Sec (Log Min), [4] For Ever Hours

//Only long Flash so varaiable a long even though not required

IntegerToCharacters ((**unsigned** **short**)usiHysHoldOffTime[3]/360); //Divide by 360 not 3600 as want pseudo decimal hours

LHemailBody[50] = Char3;

LHemailBody[51] = Char2;

LHemailBody[52] = Char1;

// usiHVAC1On24Min this time is the total time the thermostat called for ON, but is not actual Rund time

// usiHVAC1StTime[2] //Lst 24 Hr St time = Last 24 HrTotal + Last Cycle St time; count is in sconds

// usiHVAC1OnTime[2] //Lst 24 Hr On time = Last 24 HrTotal + Last Cycle On time

LHemailBody[53] = 't'; //HVAC1 On time xx.x hours

IntegerToCharacters ((usiHVAC1StTime[2] + usiHVAC1OnTime[2]) /360); //Divide by 360 not 3600 as want pseudo decimal hours

LHemailBody[54] = Char3;

LHemailBody[55] = Char2;

LHemailBody[56] = Char1;

LHemailBody[57] = 'e'; //HVAC1 Start up efficiency

//usiHVAC1StEff [5]; 0)St Eff this cycle 1) Evr St Eff 2)24Hr St Eff 3)Lst 24 St Eff Avg 4)Lst 10 cycle St Eff

IntegerToCharacters (usiHVAC1StEff [2]); //Last 24 Hr Tot On time / (Tot On + Tot Start Time)

LHemailBody[58] = Char2;

LHemailBody[59] = Char1;

LHemailBody[60] = '-';

IntegerToCharacters (usiHrsGetTime);

LHemailBody[61] = Char2;

LHemailBody[62] = Char1;

// LHemailBody[61] = 'T';

// LHemailBody[62] = 's';

LHemailBody[63] = '\0'; //End of string Null

//

// LHemailBody[63] = '\0'; //last character should already be null

//Tested since count from 0, 63 is char 64 and max that will send

usiDummy = 0;

printf("\nMessage Body is: %s\n", LHemailBody);

// CLI\_Write("\nMessage Body is: %s\n", LHemailBody);

usiDummy = 0;

} // Closing bracket for void SubjectBodyAssemble (void)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// End of ......Function Email SubjectBodyAssemble

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Function Integer to Individual Characters IntegerToCharacters

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** IntegerToCharacters (**unsigned** **short** usiNumber)

{

i = usiNumber % 10; //Give ones digit of a big number

Char1 = i + 48;

numRemain = usiNumber / 10;

i = numRemain % 10;

Char2 = i + 48;

numRemain = numRemain / 10;

i = numRemain % 10;

Char3 = i + 48;

numRemain = numRemain / 10;

i = numRemain % 10;

Char4 = i + 48;

numRemain = numRemain / 10;

i = numRemain % 10;

Char5 = i + 48;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// End of ....Function Integer to Individual Characters IntegerToCharacters

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## emailAndGetTime.c

//5-1/7-2023 LH Code for 1) Get Time 2) Send Email via http\_put to IFTTT and 3) SMTP send email code that worked for 8 years but as of Jan 27 2023 even TI can not fix.

//4-25-2023 L Hinz got http\_Put to trigger IFTTT Send Email, and still Get Time

//Feb, Mar, April near daily TI E2E post, could not solve Send Email via SNTP

//IFTTT: "If This Then That" if it continues to work with http, and not require https solves the problem of constant up dates to Comply with ever more complex Email security requirements

//4-17-2023 L Hinz merged TI http\_client example and TI email.c example, and TI Get Time Example basic WiFi and Internet code

// Then I added in function call from main for L Hinz defined Get Time and Send Email

//

// email.c - function implementation to send email

// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

/\*

\* Protocol Name - Simple Mail Transfer Protocol

\* Protocol Overview - The objective of the Simple Mail Transfer Protocol (SMTP) is to

\* transfer mail reliably and efficiently. SMTP is independent of the

\* particular transmission subsystem and requires only a reliable

\* ordered data stream channel.

\* Refer: https://www.ietf.org/rfc/rfc2821.txt

\*/

**volatile** **unsigned** **short** usiEmail1xTime2xIFTTTemail3 = 0; //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

//Same function call can be used but inside the function the specific statements are for email, get time, or IFTTT email

//I do not want to create a similar function with a different name as I do not know where else in simplelink it may be calling these functions

**#include** <emailAndGetTime.h>

**#include** "base64.h"

**#include** "config.h"

**#include** "sl\_common.h"

/\*NetApp Email protocol types \*/

**#define** SL\_NET\_APP\_SMTP\_PROTOCOL (1)

**#define** INVALID\_SOCKET\_DESC 0xFFFFFFFF

/\* Strings used to configure the Email msg in proper format \*/

**const** \_u8 smtp\_helo[] = {'H','E','L','O','\0'};

**const** \_u8 smtp\_mail\_from[]={'M','A','I','L',' ','F','R','O','M',':',' ','\0'};

**const** \_u8 smtp\_rcpt[] = {'R','C','P','T',' ','T','O',':',' ','\0'};

**const** \_u8 smtp\_data[] = "DATA";

**const** \_u8 smtp\_crlf[] = "\r\n";

**const** \_u8 smtp\_dcrlf[] = "\r\n\r\n";

**const** \_u8 smtp\_subject[] = "Subject: ";

**const** \_u8 smtp\_to[] = "To: ";

**const** \_u8 smtp\_from[] = "From: ";

/\* <CRLF>.<CRLF> Terminates the data portion \*/

**const** \_u8 smtp\_data\_end[] = {'\r','\n','.','\r','\n','\0'};

**const** \_u8 smtp\_quit[] = {'Q','U','I','T','\r','\n','\0'};

/\* Return Codes \*/

**const** \_u8 smtp\_code\_ready[] = {'2','2','0','\0'};

**const** \_u8 smtp\_ok\_reply[] = {'2','5','0','\0'};

**const** \_u8 smtp\_intermed\_reply[] = {'3','5','4','\0'};

**const** \_u8 smtp\_auth\_reply[] = {'3','3','4','\0'};

**const** \_u8 smtp\_auth\_success[] = {'2','3','5','\0'};

/\* states for smtp state machine \*/

**typedef** **enum**

{

*smtpINACTIVE* = 0,

*smtpINIT*,

*smtpHELO*,

*smtpAUTH*,

*smtpFROM*,

*smtpRCPT*,

*smtpDATA*,

*smtpMESSAGE*,

*smtpQUIT*,

*smtpERROR*

}\_SlsmtpStatus\_e;

/\* Initialize SMTP State Machine \*/

\_u16 g\_smtpStatus = *smtpINIT*;

\_u16 g\_EmailSetStatus = 0;

/\* error handling flags for smtp state machine \*/

**typedef** **enum**

{

*smtpNOERROR* = 0,

*atINIT*,

*atHELO*,

*atAUTH*,

*atFROM*,

*atRCPT*,

*atDATA*,

*atMESSAGE*,

*atQUIT*

}\_SlsmtpERROR\_e;

/\* Initialize Error handling flag \*/

\_u16 g\_smtpErrorInfo = *smtpNOERROR*;

\_i32 smtpSocket;

\_u8 g\_cmdBuf[SMTP\_BUF\_LEN];

\_u8 buf[SMTP\_BUF\_LEN];

\_u8 basekey1[BASEKEY\_LEN];

\_u8 basekey2[BASEKEY\_LEN];

\_u8 message[MAX\_MESSAGE\_LEN];

SlNetAppEmailOpt\_t g\_EmailOpt;

SlNetAppSourceEmail\_t g\_Email;

SlNetAppSourcePassword\_t g\_SourcePass;

SlNetAppDestination\_t g\_Destination;

SlNetAppEmailSubject\_t g\_Subject;

\_u8 email\_rfc[MAX\_EMAIL\_RCF\_LEN];

**static** \_i32 **\_smtpConnect**(**void**);

**static** \_i32 **\_smtpSend**(**void**);

**static** **void** **\_smtpHandleERROR**(\_u8 \* servermessage);

**static** \_i32 **\_sendSMTPCommand**(\_i32 socket, \_u8 \* cmd, \_u8 \* cmdparam, \_u8 \* respBuf);

**static** **void** **\_generateBase64Key**(\_u8 \* basekey1, \_u8 \* basekey2);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Top files of Send Email example

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//LH These 3 defines already in email.c, so copy from main not needed here

//#include "email.h"

//#include "config.h"

//#include "sl\_common.h"

**#define** APPLICATION\_VERSION "1.3.0"

**#define** SL\_STOP\_TIMEOUT 0xFF

/\*

\* GLOBAL VARIABLES -- Start

\*/

\_u32 g\_Status = 0;

/\*

\* GLOBAL VARIABLES -- End

\*/

//volatile unsigned short usiEmailDummy = 0;

/\*

\* STATIC FUNCTION DEFINITIONS -- Start

\*/

**static** \_i32 **establishConnectionWithAP**();

**static** \_i32 **disconnectFromAP**();

**static** \_i32 **configureSimpleLinkToDefaultState**();

**static** \_i32 **initializeAppVariables**();

**static** **void** **displayBanner**();

**static** \_i32 **setEmail**();

**static** \_i32 **sendEmail**();

/\*

\* STATIC FUNCTION DEFINITIONS -- End

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//End of .... Top files of Send Email example

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Top functions and variables from TI http\_client example

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* main.c - HTTP Client application

\*

\* Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

\* Application Name - HTTP Client

\* Application Overview - This sample application demonstrates how to use

\* HTTP Client (In Minimum mode) API for HTTP based

\* application development.

\* This application explain user to how to:

\* 1. Connect to an access point

\* 2. Connect to a HTTP Server with and without proxy

\* 3. Do POST, GET, PUT and DELETE

\* 4. Parse JSON data using “Jasmine JSON Parser”

\*

\* Note: To use HTTP Client in minimum mode, user need to compile library (http\_lib)

\* with HTTPCli\_LIBTYPE\_MIN option.

\*

\* HTTP Client (minimal) library supports synchronous mode, redirection

\* handling, chunked transfer encoding support, proxy support and TLS

\* support (for SimpleLink Only. TLS on other platforms are disabled)

\*

\* HTTP Client (Full) library supports all the features of the minimal

\* library + asynchronous mode and content handling support +

\* TLS support and requires RTOS support and not supported on CC3100.

\*

\* Application Details - http://processors.wiki.ti.com/index.php/CC31xx\_HTTP\_Client

\* doc\examples\http\_client.pdf

\*/

**#include** "simplelink.h"

//#include "sl\_common.h" //LH comment out 4-17-2023,Already in email section line 26

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

/\* HTTP Client lib include \*/

**#include** <http/client/httpcli.h>

//4-21-2023 LH Added next 3 include to see if solve No or unrecognized network configuration

//#include "httpcli.h"

//#include "httpstr.h"

//#include "ssock.h"

/\* JSON Parser include \*/

**#include** "jsmn.h"

//#define APPLICATION\_VERSION "1.3.0" //LH comment out 4-17-2023, Already defined line 119

//#define SL\_STOP\_TIMEOUT 0xFF //LH comment out 4-17-2023,Already defined 945 and 1709

/\*

\* HTTP request parameters. These may change as depending upon the server

\*/

//#define POST\_REQUEST\_URI "/post" //4-20-2023 not used in IFTTT put email application

//#define POST\_REQUEST\_URI "/trigger/SendEmailV2/with/key/xxxx-My-IFTTT-Unique-Key-xx?value1=97.8"

//#define POST\_DATA "{\n\"name\":\"xyz\",\n\"address\":\n{\n\"plot#\":12,\n\"street\":\"abc\",\n\"city\":\"ijk\"\n},\n\"age\":30\n}"

//The long string above with name, address, etc is TI example Post that works

//#define POST\_DATA "{}" //Try with Post Data blank for "http://maker.ifttt.com"

//#define POST\_DATA "{/trigger/SendEmailV2/with/key/xxxx-My-IFTTT-Unique-Key-xx?value1=97.8}"

//#define POST\_DATA "{/SendEmailV2/with/key/xxx-My-Unique-Key-xx?value1=97.8}"

//#define DELETE\_REQUEST\_URI "/delete" //4-20-2023 not used in IFTTT put email application

//#define PUT\_REQUEST\_URI "/put"

//#define PUT\_REQUEST\_URI "/trigger/SendEmailV2/with/key/xxxx-My-IFTTT-Unique-Key-xx?value1=GenAmmeter&value2=46.78"

//#define PUT\_REQUEST\_URI "/trigger/SendEmailV2/with/key/xxx-My-Unique-Key-xx?value1=GenAmmeter&value2=AmpGnPoHv-M01D22-T1241-GM01D10-UON00.0-Vb683-T056m55h060L50H075-Body-1L045M124-2L158M259-Gt000m49-Pa088m090t015m959-Ha034M36t071m097"

//After changed lengths in http\_lib, httpcli.c this long string works

**#define** PUT\_DATA "PUT request."

//#define GET\_REQUEST\_URI "/get" //4-20-2023 not used in IFTTT put email application

//#define HOST\_NAME "httpbin.org"

//Later learnt that should not include http:// ....Simplelink will add this, so duplication will not work

//\*\*\*\* DO NOT INCLUDE http:// in front of the host name

**#define** HOST\_NAME "maker.ifttt.com"

**#define** HOST\_PORT 80

**#define** PROXY\_IP 0xBA5FB660 //4-1-2023 Example had this, LH commented out, example still works

**#define** PROXY\_PORT <proxy\_port> //4-1-2023 Example had this, LH commented out, example still works, test to get my http to work

**#define** READ\_SIZE 1450

**#define** MAX\_BUFF\_SIZE 1460

**#define** SPACE 32 //TI value

// 4-23-2023 Comment all out as conflict with config.h, added ones missing to config.h

// This removes a conflict compile error

///\* Application specific status/error codes \*/

//typedef enum{

//// DEVICE\_NOT\_IN\_STATION\_MODE = -0x7D0, /\* Choosing this number to avoid overlap with host-driver's error codes \*/

// INVALID\_HEX\_STRING = DEVICE\_NOT\_IN\_STATION\_MODE - 1,

//// TCP\_RECV\_ERROR = INVALID\_HEX\_STRING - 1,

//// TCP\_SEND\_ERROR = TCP\_RECV\_ERROR - 1, //LH 4-17-2023 comment out as Get Time an Email already had these

// FILE\_NOT\_FOUND\_ERROR = TCP\_SEND\_ERROR - 1,

// INVALID\_SERVER\_RESPONSE = FILE\_NOT\_FOUND\_ERROR - 1,

// FORMAT\_NOT\_SUPPORTED = INVALID\_SERVER\_RESPONSE - 1,

// FILE\_WRITE\_ERROR = FORMAT\_NOT\_SUPPORTED - 1,

// INVALID\_FILE = FILE\_WRITE\_ERROR - 1,

//

//// STATUS\_CODE\_MAX = -0xBB8

//}e\_AppStatusCodes;

/\*

\* GLOBAL VARIABLES -- Start

\*/

\_u32 g\_Status;

\_u32 g\_DestinationIP;

\_u32 g\_BytesReceived; /\* variable to store the file size \*/

\_u8 g\_buff[MAX\_BUFF\_SIZE+1];

\_i32 g\_SockID = 0;

/\*

\* GLOBAL VARIABLES -- End

\*/

/\*

\* STATIC FUNCTION DEFINITIONS -- Start

\*/

//static \_i32 establishConnectionWithAP(); //LH comment out 4-17-2023 as already line 961

//static \_i32 configureSimpleLinkToDefaultState(); //LH comment out 4-17-2023 as already line 963

//static \_i32 initializeAppVariables(); //LH comment out 4-17-2023 as already line 965

//static void displayBanner(); //LH comment out 4-17-2023 as already line 966

**static** \_i32 **ConnectToHTTPServer**(HTTPCli\_Handle httpClient);

//static \_i32 HTTPPostMethod(HTTPCli\_Handle httpClient); //4-18-2023, not used for IFTTT http-put to send email

//static \_i32 HTTPDeleteMethod(HTTPCli\_Handle httpClient); //4-18-2023, not used for IFTTT http-put to send email

**static** \_i32 **HTTPPutMethod**(HTTPCli\_Handle httpClient);

//static \_i32 HTTPGetMethod(HTTPCli\_Handle httpClient); //4-18-2023, not used for IFTTT http-put to send email

**static** \_i32 **readResponse**(HTTPCli\_Handle httpClient);

**static** **void** **FlushHTTPResponse**(HTTPCli\_Handle httpClient);

**static** \_i32 **ParseJSONData**(\_i8 \*ptr);

/\*

\* STATIC FUNCTION DEFINITIONS -- End

\*/

//char myString[5] = {'\0'};

//char myString[5] = {'M','i','k','e','\0'};

//char myString[5] = "Mike";

//char PUT\_REQUEST\_URI[5] = "Mike"; //This compiles OK

//char PUT\_REQUEST\_URI[211] = "/trigger/SendEmailV2/with/key/xxxx-My-IFTTT-Unique-Key-xx?value1=GenAmmeter&value2=AmpGnPoHv-M01D22-T1241-GM01D10-UON00.0-Vb683-T056m55h060L50H075-Body-1L045M124-2L158M259-Gt000m49-Pa088m090t015m959-Ha034M36t071m097";

//210 Characters, like above example string need to be 1 char longer for end NULL

**char** PUT\_REQUEST\_URI[214] = "/trigger/SendEmailV2/with/key/xxxx-My-IFTTT-Unique-Key-xx?value1=LHProd15CharNam&value2=1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567-ZX";

//char LHemailSubject[64] = "AmpGnPoHv-M01D22-T1241-GM01D10-UON00.0-Vb683-T056m55h060L50H075";

//char LHemailBody[64] = "1L045M124-2L158M259-Gt000m49-Pa088m090t015m959-Ha034M36t071m097";

//Select Product Name or Add a new Product name, must be exactly 15 characters, no less, no more

//char LHProd15CharNam [16] = "AmetrGenPolHvac"; //15 characters plus null

//char LHProd15CharNam [16] = "Hobby-Prototype";

//char LHProd15CharNam [16] = "LwGrAtticFnHVAC";

**char** LHProd15CharNam [16] = "UpAttic2FnsHVAC";

//char LHProd15CharNam [16] = "StdyTherHystCtr";

//char LHProd15CharNam [16] = "StrFnGrdSaWatSw";

//char LHProd15CharNam [16] = "SolarController";

//char LHProd15CharNam [16] = "Water-FL-Sentry";

**volatile** **unsigned** **short** usiURIarrayCt = 0;

**volatile** **unsigned** **short** usiDataarrayCt = 0;

**volatile** **unsigned** **short** usiEmailDummy = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//End of ...............Top functions and variables from TI http\_client example

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Copied main in TI http\_client example, changed to an LHIFTTTSendEamil

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int** **LHIFTTTSendEamil**() // 4-18-2023

{

usiEmailDummy = 0;

usiEmail1xTime2xIFTTTemail3 = 3; //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

/\*

\* Application's entry point From Main of http\_client

\*/

//int main(int argc, char\*\* argv)

//{

// This works

// PUT\_REQUEST\_URI[200] = 'X';

// PUT\_REQUEST\_URI[201] = 'Y';

// PUT\_REQUEST\_URI[202] = 'Z';

//Load Product Name String into 214 character URI

usiURIarrayCt = 60; //value= = is character 61

**for** (usiDataarrayCt = 0; usiDataarrayCt <=14; usiDataarrayCt++){ //So 15 times loop as don't want to load the null

PUT\_REQUEST\_URI[usiURIarrayCt] = LHProd15CharNam [usiDataarrayCt];

usiURIarrayCt++;

}

//Load Subject Data String into 214 character URI

usiURIarrayCt = 83; //value= = is character 82

**for** (usiDataarrayCt = 0; usiDataarrayCt <=62; usiDataarrayCt++){ //So 63 times loop as don't want to load the null

PUT\_REQUEST\_URI[usiURIarrayCt] = LHemailSubject [usiDataarrayCt];

usiURIarrayCt++;

}

//Load Body Data String into 214 character URI

PUT\_REQUEST\_URI[usiURIarrayCt] = '-'; //Space Between Subject and Body

usiURIarrayCt++;

**for** (usiDataarrayCt = 0; usiDataarrayCt <=62; usiDataarrayCt++){ //So 63 times loop as don't want to load the null

PUT\_REQUEST\_URI[usiURIarrayCt] = LHemailBody [usiDataarrayCt];

usiURIarrayCt++;

}

//PUT\_REQUEST\_URI = "/trigger/SendEmailV2/with/key/xxxx-My-IFTTT-Unique-Key-xx?value1=GenAmmeter&value2=AmpGnPoHv-M01D22-T1241-GM01D10-UON00.0-Vb683-T056m55h060L50H075-Body-1L045M124-2L158M259-Gt000m49-Pa088m090t015m959-Ha034M36t071m097";

\_i32 retVal = -1;

HTTPCli\_Struct httpClient;

retVal = initializeAppVariables();

ASSERT\_ON\_ERROR(retVal);

/\* Stop WDT and initialize the system-clock of the MCU \*/

// stopWDT();

// initClk();

/\* Configure command line interface \*/

CLI\_Configure();

displayBanner();

/\*

\* Following function configures the device to default state by cleaning

\* the persistent settings stored in NVMEM (viz. connection profiles &

\* policies, power policy etc)

\*

\* Applications may choose to skip this step if the developer is sure

\* that the device is in its default state at start of application

\*

\* Note that all profiles and persistent settings that were done on the

\* device will be lost

\*/

retVal = configureSimpleLinkToDefaultState();

**if**(retVal < 0)

{

**if** (*DEVICE\_NOT\_IN\_STATION\_MODE* == retVal)

{

CLI\_Write(" Failed to configure the device in its default state \n\r");

}

LOOP\_FOREVER();

}

CLI\_Write(" Device is configured in default state \n\r");

/\*

\* Initializing the CC3100 device

\* Assumption is that the device is configured in station mode already

\* and it is in its default state

\*/

retVal = sl\_Start(0, 0, 0);

**if** ((retVal < 0) ||

(*ROLE\_STA* != retVal) )

{

CLI\_Write(" Failed to start the device \n\r");

LOOP\_FOREVER();

}

CLI\_Write(" Device started as STATION \n\r");

/\* Connecting to WLAN AP \*/

retVal = establishConnectionWithAP();

**if**(retVal < 0)

{

CLI\_Write(" Failed to establish connection w/ an AP \n\r");

LOOP\_FOREVER();

}

CLI\_Write(" Connection established w/ AP and IP is acquired \n\r");

/\* Connect to HTTP server \*/

retVal = ConnectToHTTPServer(&httpClient);

**if**(retVal < 0)

{

LOOP\_FOREVER();

}

CLI\_Write("\n\r");

// CLI\_Write(" HTTP Post Test Begin:\n\r");

// retVal = HTTPPostMethod(&httpClient);

// if(retVal < 0)

// {

// CLI\_Write(" HTTP Post Test failed.\n\r");

// }

// CLI\_Write(" HTTP Post Test Completed Successfully\n\r");

//

// CLI\_Write("\n\r");

// CLI\_Write(" HTTP Delete Test Begin:\n\r");

// retVal = HTTPDeleteMethod(&httpClient);

//

// if(retVal < 0)

// {

// CLI\_Write(" HTTP Delete Test failed.\n\r");

// }

// CLI\_Write(" HTTP Delete Test Completed Successfully\n\r");

CLI\_Write("\n\r");

CLI\_Write(" HTTP Put Test Begin:\n\r");

retVal = HTTPPutMethod(&httpClient);

**if**(retVal < 0)

{

CLI\_Write(" HTTP Put Test failed.\n\r");

}

CLI\_Write(" HTTP Put Test Completed Successfully\n\r");

CLI\_Write("\n\r");

// CLI\_Write(" HTTP Get Test Begin:\n\r");

// retVal = HTTPGetMethod(&httpClient);

// if(retVal < 0)

// {

// CLI\_Write(" HTTP Get Test failed.\n\r");

// }

// CLI\_Write(" HTTP Get Test Completed Successfully\n\r");

// CLI\_Write("\n\r");

/\* Stop the CC3100 device \*/

retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

**if**(retVal < 0)

{

LOOP\_FOREVER();

}

**return** SUCCESS;

} // Closing bracket for int LHIFTTTSendEamil()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//End of ......Copied main in TI http\_client example, changed to an LHIFTTTSendEamil

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*!

\brief This function sets the email parameters

\param[in] command - Command send for processing

\param[in] pValueLen - Length of data to be processed

\param[in] pValue - Data to be processed

\return 0 for success, -1 otherwise

\note

\warning

\*/

\_i32 **sl\_NetAppEmailSet**(\_u8 command ,\_u8 pValueLen,

\_u8 \*pValue)

{

SlNetAppEmailOpt\_t\* pEmailOpt = 0;

SlNetAppSourceEmail\_t\* pSourceEmail = NULL;

SlNetAppSourcePassword\_t\* pSourcePassword = NULL;

SlNetAppDestination\_t\* pDestinationEmail = NULL;

SlNetAppEmailSubject\_t\* pSubject = NULL;

**switch** (command)

{

**case** NETAPP\_ADVANCED\_OPT:

pEmailOpt = (SlNetAppEmailOpt\_t\*)pValue;

g\_EmailOpt.Port = pEmailOpt->Port;

g\_EmailOpt.Family = pEmailOpt->Family;

g\_EmailOpt.SecurityMethod = pEmailOpt->SecurityMethod;

g\_EmailOpt.SecurityCypher = pEmailOpt->SecurityCypher;

g\_EmailOpt.Ip = pEmailOpt->Ip;

g\_EmailSetStatus+=1;

**break**;

**case** NETAPP\_SOURCE\_EMAIL:

pSourceEmail = (SlNetAppSourceEmail\_t\*)pValue;

pal\_Memset(g\_Email.Username, '\0', MAX\_USERNAME\_LEN);

pal\_Memcpy(g\_Email.Username, pSourceEmail->Username, pValueLen);

g\_EmailSetStatus+=2;

**break**;

**case** NETAPP\_PASSWORD:

pSourcePassword = (SlNetAppSourcePassword\_t\*)pValue;

pal\_Memset(g\_SourcePass.Password, '\0', MAX\_PASSWORD\_LEN);

pal\_Memcpy(g\_SourcePass.Password, pSourcePassword->Password, pValueLen);

g\_EmailSetStatus+=4;

**break**;

**case** NETAPP\_DEST\_EMAIL:

pDestinationEmail=(SlNetAppDestination\_t\*)pValue;

pal\_Memset(g\_Destination.Email, '\0', MAX\_DEST\_EMAIL\_LEN);

pal\_Memcpy(g\_Destination.Email, pDestinationEmail->Email, pValueLen);

g\_EmailSetStatus+=8;

**break**;

**case** NETAPP\_SUBJECT:

pSubject=(SlNetAppEmailSubject\_t\*)pValue;

pal\_Memset(g\_Subject.Value, '\0', MAX\_SUBJECT\_LEN);

pal\_Memcpy(g\_Subject.Value, pSubject->Value, pValueLen);

g\_EmailSetStatus+=16;

**break**;

**case** NETAPP\_MESSAGE:

**if**(pValueLen > (MAX\_MESSAGE\_LEN - 1))

{

ASSERT\_ON\_ERROR(*EMAIL\_SET\_INVALID\_MESSAGE*);

}

pal\_Memset(message, '\0', MAX\_MESSAGE\_LEN);

pal\_Memcpy(message ,pValue, pValueLen);

**break**;

**default**:

CLI\_Write((\_u8\*)"\n\rError:Default case\n\r");

ASSERT\_ON\_ERROR(*EMAIL\_SET\_INVALID\_CASE*);

}

**return** SUCCESS;

}

/\*!

\brief Create a secure socket and connects to SMTP server

\param[in] none

\return 0 if success and negative in case of error

\note

\warning

\*/

\_i32 **sl\_NetAppEmailConnect**()

{

**if** (!(g\_EmailSetStatus >= 0x07))

{

CLI\_Write((\_u8\*)"\n\rError:Email and Subject is not configured\n\r");

ASSERT\_ON\_ERROR(*EMAIL\_CONNECT\_INVALID\_CONFIURATION*);

}

**return** \_smtpConnect();

}

/\*!

\brief Check the connection status and sends the Email

\param[in] none

\return 0 if success otherwise -1

\note

\warning

\*/

\_i32 **sl\_NetAppEmailSend**()

{

\_i32 retVal = -1;

retVal = \_smtpSend();

**if**(retVal < 0)

sl\_Close(smtpSocket);

**return** retVal;

}

/\*!

\brief Creates a secure socket and connects to SMTP server

\param[in] none

\return 0 if success and negative in case of error

\note

\warning

\*/

**static** \_i32 **\_smtpConnect**(**void**)

{

SlSockAddrIn\_t LocalAddr;

SlTimeval\_t tTimeout;

\_i32 cipher = 0;

\_i32 LocalAddrSize = 0;

\_i8 method = 0;

\_i32 Status = 0;

LocalAddr.sin\_family = g\_EmailOpt.Family;

LocalAddr.sin\_port = sl\_Htons(g\_EmailOpt.Port);

LocalAddr.sin\_addr.s\_addr = sl\_Htonl(g\_EmailOpt.Ip);

LocalAddrSize = **sizeof**(SlSockAddrIn\_t);

/\* If TLS is required \*/

**if**(g\_EmailOpt.SecurityMethod <= 5)

{

/\* Create secure socket \*/

smtpSocket = sl\_Socket(SL\_AF\_INET, SL\_SOCK\_STREAM, SL\_SEC\_SOCKET);

ASSERT\_ON\_ERROR(smtpSocket);

tTimeout.tv\_sec = 10;

tTimeout.tv\_usec = 90000;

Status = sl\_SetSockOpt(smtpSocket, SOL\_SOCKET, SL\_SO\_RCVTIMEO,

&tTimeout, **sizeof**(SlTimeval\_t));

ASSERT\_ON\_ERROR(Status);

method = g\_EmailOpt.SecurityMethod;

cipher = g\_EmailOpt.SecurityCypher;

/\* Set Socket Options that were just defined \*/

Status = sl\_SetSockOpt(smtpSocket, SL\_SOL\_SOCKET, SL\_SO\_SECMETHOD,

&method, **sizeof**(method));

**if**( Status < 0 )

{

sl\_Close(smtpSocket);

ASSERT\_ON\_ERROR(Status);

}

Status = sl\_SetSockOpt(smtpSocket, SL\_SOL\_SOCKET, SL\_SO\_SECURE\_MASK,

&cipher, **sizeof**(cipher));

**if**( Status < 0 )

{

sl\_Close(smtpSocket);

ASSERT\_ON\_ERROR(Status);

}

}

/\* If no TLS required \*/

**else**

{

/\* Create socket \*/

smtpSocket = sl\_Socket(SL\_AF\_INET, SL\_SOCK\_STREAM, SL\_IPPROTO\_TCP);

ASSERT\_ON\_ERROR(smtpSocket);

}

/\* connect to socket \*/

Status = sl\_Connect(smtpSocket, (SlSockAddr\_t \*)&LocalAddr, LocalAddrSize);

**if**((Status < 0) && (SL\_ESECSNOVERIFY != Status))

{

ASSERT\_ON\_ERROR(Status);

}

**return** SUCCESS;

}

/\*!

\brief This function will send the email

\param[in] none

\return 0 if success otherwise -1

\note

\warning

\*/

**static** \_i32 **\_smtpSend**(**void**)

{

\_i16 exit = 0;

\_i32 retVal = -1;

/\* If socket has been opened, check for acknowledge from SMTP server \*/

**while**(exit == 0)

{

**switch**(g\_smtpStatus)

{

/\*

\* An SMTP session is initiated when a client opens a connection to

\* a server and the server responds with an opening message.

\*/

**case** *smtpINIT*:

/\* Create buffer, Read so we can check for 220 'OK' from server \*/

pal\_Memset(buf, 0, SMTP\_BUF\_LEN);

retVal = sl\_Recv(smtpSocket, buf, **sizeof**(buf), 0);

**if**(retVal <= 0)

{

ASSERT\_ON\_ERROR(*TCP\_RECV\_ERROR*);

}

/\* If buffer has 220, set state to HELO \*/

**if**(buf[0] == smtp\_code\_ready[0] &&

buf[1] == smtp\_code\_ready[1] &&

buf[2] == smtp\_code\_ready[2])

{

g\_smtpStatus = *smtpHELO*;

}

/\* Else error, set state to ERROR \*/

**else**

{

g\_smtpStatus = *smtpERROR*;

g\_smtpErrorInfo = *atINIT*;

}

**break**;

/\*

\* The client normally sends the EHLO command to the

\* server, indicating the client's identity. In addition to opening the

\* session, use of EHLO indicates that the client is able to process

\* service extensions and requests that the server provide a list of the

\* extensions it supports.

\*/

**case** *smtpHELO*:

retVal = \_sendSMTPCommand(smtpSocket, "HELO localhost", NULL, buf);

ASSERT\_ON\_ERROR(retVal);

/\* If response has 250, set state to AUTH \*/

**if**(buf[0] == smtp\_ok\_reply[0] &&

buf[1] == smtp\_ok\_reply[1] &&

buf[2] == smtp\_ok\_reply[2])

{

g\_smtpStatus = *smtpAUTH*;

}

/\* Else error, set state to ERROR \*/

**else**

{

g\_smtpStatus = *smtpERROR*;

g\_smtpErrorInfo = *atHELO*;

}

**break**;

/\* The client sends the AUTH command with SASL mechanism to use with \*/

**case** *smtpAUTH*:

/\*

\* SASL PLain - Authentication with server username and password

\* Refer - http://www.ietf.org/rfc/rfc4616.txt

\*/

\_generateBase64Key((\_u8\*)g\_Email.Username, basekey1);

\_generateBase64Key((\_u8\*)g\_SourcePass.Password, basekey2);

/\* Send request to server for authentication \*/

retVal = \_sendSMTPCommand(smtpSocket, "AUTH LOGIN", NULL, buf);

ASSERT\_ON\_ERROR(retVal);

/\* If response has 334, give username in base64 \*/

**if**(buf[0] == smtp\_auth\_reply[0] &&

buf[1] == smtp\_auth\_reply[1] &&

buf[2] == smtp\_auth\_reply[2])

{

retVal = \_sendSMTPCommand(smtpSocket, basekey1, NULL, buf);

ASSERT\_ON\_ERROR(retVal);

/\* If response has 334, give password in base64 \*/

**if**(buf[0] == smtp\_auth\_reply[0] &&

buf[1] == smtp\_auth\_reply[1] &&

buf[2] == smtp\_auth\_reply[2])

{

retVal = \_sendSMTPCommand(smtpSocket, basekey2, NULL, buf);

ASSERT\_ON\_ERROR(retVal);

}

}

**if**(buf[0] == smtp\_auth\_success[0] &&

buf[1] == smtp\_auth\_success[1] &&

buf[2] == smtp\_auth\_success[2])

{

/\* Authentication was successful, set state to FROM \*/

g\_smtpStatus = *smtpFROM*;

}

/\* Else error, set state to ERROR \*/

**else**

{

g\_smtpStatus = *smtpERROR*;

g\_smtpErrorInfo = *atAUTH*;

}

**break**;

/\*

\* The SMTP transaction starts with a MAIL command which includes the

\* sender information

\* MAIL FROM:<reverse-path> [SP <mail-parameters> ] <CRLF>

\*/

**case** *smtpFROM*:

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_mail\_from,

(\_u8 \*)g\_Email.Username, buf);

ASSERT\_ON\_ERROR(retVal);

/\* If response has 250, set state to RCPT \*/

**if**(buf[0] == smtp\_ok\_reply[0] &&

buf[1] == smtp\_ok\_reply[1] &&

buf[2] == smtp\_ok\_reply[2])

{

g\_smtpStatus = *smtpRCPT*;

}

**else**

{

pal\_Memset(email\_rfc,'\0',MAX\_EMAIL\_RCF\_LEN);

pal\_Memcpy(email\_rfc,"<",1);

pal\_Memcpy(&email\_rfc[1], g\_Email.Username,

pal\_Strlen(g\_Email.Username));

pal\_Memcpy(&email\_rfc[1+pal\_Strlen(g\_Email.Username)],">",1);

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_mail\_from,

email\_rfc, buf);

ASSERT\_ON\_ERROR(retVal);

**if**(buf[0] == smtp\_ok\_reply[0] &&

buf[1] == smtp\_ok\_reply[1] &&

buf[2] == smtp\_ok\_reply[2])

{

g\_smtpStatus = *smtpRCPT*;

}

**else**

{

g\_smtpStatus = *smtpERROR*;

g\_smtpErrorInfo = *atFROM*;

}

}

**break**;

/\* Send the destination email to the smtp server

\* RCPT TO:<forward-path> [ SP <rcpt-parameters> ] <CRLF>

\*/

**case** *smtpRCPT*:

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_rcpt,

(\_u8 \*)g\_Destination.Email, buf);

ASSERT\_ON\_ERROR(retVal);

/\* If response has 250, set state to DATA \*/

**if**(buf[0] == smtp\_ok\_reply[0] &&

buf[1] == smtp\_ok\_reply[1] &&

buf[2] == smtp\_ok\_reply[2])

{

g\_smtpStatus = *smtpDATA*;

}

**else**

{

pal\_Memset(email\_rfc,'\0',MAX\_EMAIL\_RCF\_LEN);

pal\_Memcpy(email\_rfc,"<",1);

pal\_Memcpy(&email\_rfc[1], g\_Destination.Email,

pal\_Strlen(g\_Destination.Email));

pal\_Memcpy(&email\_rfc[1+pal\_Strlen(g\_Destination.Email)],">",1);

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_rcpt,email\_rfc, buf);

ASSERT\_ON\_ERROR(retVal);

/\* If response has 250, set state to DATA \*/

**if**(buf[0] == smtp\_ok\_reply[0] &&

buf[1] == smtp\_ok\_reply[1] &&

buf[2] == smtp\_ok\_reply[2])

{

g\_smtpStatus = *smtpDATA*;

}

**else**

{

g\_smtpStatus = *smtpERROR*;

g\_smtpErrorInfo = *atRCPT*;

}

}

**break**;

/\*Send the "DATA" message to the server, indicates client is ready

\* to construct the body of the email

\* DATA <CRLF>

\*/

**case** *smtpDATA*:

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_data, NULL, buf);

ASSERT\_ON\_ERROR(retVal);

/\* If Response has 250, set state to MESSAGE \*/

**if**(buf[0] == smtp\_intermed\_reply[0] &&

buf[1] == smtp\_intermed\_reply[1] && buf[2] == smtp\_intermed\_reply[2])

{

g\_smtpStatus = *smtpMESSAGE*;

}

**else**

{

g\_smtpStatus = *smtpERROR*;

g\_smtpErrorInfo = *atDATA*;

}

**break**;

**case** *smtpMESSAGE*:

/\* Send actual Message, preceded by FROM, TO and Subject \*/

/\* Start with E-Mail's "Subject:" field \*/

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_subject,

(\_u8 \*)g\_Subject.Value, NULL);

ASSERT\_ON\_ERROR(retVal);

/\* Add E-mail's "To:" field \*/

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_to,

(\_u8 \*)g\_Destination.Email, NULL);

ASSERT\_ON\_ERROR(retVal);

/\* Add E-mail's "From:" field \*/

retVal = \_sendSMTPCommand(smtpSocket, (\_u8 \*)smtp\_from,

(\_u8 \*)g\_Email.Username, NULL);

ASSERT\_ON\_ERROR(retVal);

/\* Send CRLF \*/

retVal = sl\_Send(smtpSocket,smtp\_crlf,pal\_Strlen(smtp\_crlf),0);

**if**(retVal <= 0)

ASSERT\_ON\_ERROR(*TCP\_SEND\_ERROR*);

/\* Send body of message \*/

retVal = \_sendSMTPCommand(smtpSocket, message, NULL, NULL);

ASSERT\_ON\_ERROR(retVal);

/\* End Message \*/

retVal = \_sendSMTPCommand(smtpSocket,(\_u8 \*)smtp\_data\_end,NULL,buf);

ASSERT\_ON\_ERROR(retVal);

/\* Server will send 250 for successful. Move into QUIT state. \*/

**if**(buf[0] == smtp\_ok\_reply[0] && buf[1] == smtp\_ok\_reply[1] &&

buf[2] == smtp\_ok\_reply[2])

{

/\* Disconnect from server by sending QUIT command \*/

retVal = sl\_Send(smtpSocket,smtp\_quit,pal\_Strlen(smtp\_quit),0);

**if**(retVal <= 0)

ASSERT\_ON\_ERROR(*TCP\_SEND\_ERROR*);

/\* Close socket and reset \*/

retVal = sl\_Close(smtpSocket);

smtpSocket = INVALID\_SOCKET\_DESC;

/\* Reset the state machine \*/

g\_smtpStatus = *smtpINIT*;

exit = 1;

}

**else**

{

g\_smtpStatus = *smtpERROR*;

g\_smtpErrorInfo = *atMESSAGE*;

}

**break**;

**case** *smtpERROR*:

/\* Error Handling for SMTP \*/

\_smtpHandleERROR(buf);

/\* Close socket and reset \*/

retVal = sl\_Close(smtpSocket);

/\*Reset the state machine \*/

g\_smtpStatus = *smtpINIT*;

ASSERT\_ON\_ERROR(*SMTP\_ERROR*);

**default**:

ASSERT\_ON\_ERROR(*SMTP\_INVALID\_CASE*);

}

}

**return** SUCCESS;

}

/\*!

\brief This function converts the string to Base64 format needed for

authentication

\param[in] input - pointer to string to be converted

\param[out] basekey1 - Pointer to string for base64 converted output

\return None

\note

\warning

\*/

**static** **void** **\_generateBase64Key**(\_u8 \*input, \_u8 \*basekey1)

{

\_u8 \*pIn = input;

/\* Convert to base64 format \*/

ConvertToBase64(basekey1, (**void** \*)pIn, pal\_Strlen(input));

}

/\*!

\brief Performs Error Handling for SMTP State Machine

\param[in] servermessage - server response message

\return None

\note

\warning

\*/

**static** **void** **\_smtpHandleERROR**(\_u8 \*servermessage)

{

/\* Errors are handled using flags set in the smtpStateMachine \*/

**switch**(g\_smtpErrorInfo)

{

**case** *atINIT*:

/\* Server connection could not be established \*/

CLI\_Write((\_u8\*)"Server connection error.\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

**case** *atHELO*:

/\* Server did not accept the HELO command from server \*/

CLI\_Write((\_u8\*)"Server did not accept HELO:\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

**case** *atAUTH*:

/\* Server did not accept authorization credentials \*/

CLI\_Write((\_u8\*)"Authorization unsuccessful, ");

CLI\_Write((\_u8\*)"check username/password.\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

**case** *atFROM*:

/\* Server did not accept source email. \*/

CLI\_Write((\_u8\*)"Email of sender not accepted by server.\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

**case** *atRCPT*:

/\* Server did not accept destination email \*/

CLI\_Write((\_u8\*)"Email of recipient not accepted by server.\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

**case** *atDATA*:

/\* 'DATA' command to server was unsuccessful \*/

CLI\_Write((\_u8\*)"smtp 'DATA' command not accepted by server.\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

**case** *atMESSAGE*:

/\* Message body could not be sent to server \*/

CLI\_Write((\_u8\*)"Email Message was not accepted by the server.\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

**case** *atQUIT*:

/\* Message could not be finalized \*/

CLI\_Write((\_u8\*)"Connection could not be properly closed.");

CLI\_Write((\_u8\*)" Message not sent.\r\n");

CLI\_Write((\_u8\*)servermessage);

**break**;

}

}

/\*!

\brief Sends the SMTP command and receives the server response.

If cmd and cmd parameter are NULL, it will only send <CR><LF>

\param[in] socket - Socket Descriptor

\param[in] cmd - command to be send to server

\param[in] cmdparam - command parameter to be send

\param[in] respBuf - Pointer to buffer for SMTP server response

\return None

\note

\warning

\*/

**static** \_i32 **\_sendSMTPCommand**(\_i32 socket, \_u8 \*cmd, \_u8 \* cmdparam, \_u8 \*respBuf)

{

\_i16 sendLen = 0;

\_i32 retVal = -1;

pal\_Memset(g\_cmdBuf, 0, **sizeof**(g\_cmdBuf));

**if**(cmd != NULL)

{

sendLen = pal\_Strlen(cmd);

pal\_Memcpy(g\_cmdBuf,cmd,pal\_Strlen(cmd));

}

**if**(cmdparam != NULL)

{

pal\_Memcpy(&g\_cmdBuf[sendLen], cmdparam, pal\_Strlen(cmdparam));

sendLen += pal\_Strlen(cmdparam);

}

pal\_Memcpy(&g\_cmdBuf[sendLen], smtp\_crlf, pal\_Strlen(smtp\_crlf));

sendLen += pal\_Strlen(smtp\_crlf);

retVal = sl\_Send(socket, g\_cmdBuf,sendLen,0);

**if**(retVal <= 0)

ASSERT\_ON\_ERROR(*TCP\_SEND\_ERROR*);

**if**(respBuf != NULL)

{

pal\_Memset(respBuf,0,SMTP\_BUF\_LEN);

retVal = sl\_Recv(socket, respBuf,SMTP\_BUF\_LEN,0);

**if**(retVal <= 0)

ASSERT\_ON\_ERROR(*TCP\_RECV\_ERROR*);

}

**return** SUCCESS;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// End of Original Email file- start of Email cut and paste from Main

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

////LH These 3 defines already in email.c, so copy from main not needed here

////#include "email.h"

////#include "config.h"

////#include "sl\_common.h"

//

//#define APPLICATION\_VERSION "1.3.0"

//

//#define SL\_STOP\_TIMEOUT 0xFF

//

///\*

// \* GLOBAL VARIABLES -- Start

// \*/

//\_u32 g\_Status = 0;

///\*

// \* GLOBAL VARIABLES -- End

// \*/

//

////volatile unsigned short usiEmailDummy = 0;

//

///\*

// \* STATIC FUNCTION DEFINITIONS -- Start

// \*/

//static \_i32 establishConnectionWithAP();

//static \_i32 disconnectFromAP();

//static \_i32 configureSimpleLinkToDefaultState();

//

//static \_i32 initializeAppVariables();

//static void displayBanner();

//

//static \_i32 setEmail();

//static \_i32 sendEmail();

///\*

// \* STATIC FUNCTION DEFINITIONS -- End

// \*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Start of function int LHSendEmail()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* Application's entry point

\*/

//void LHSendEmail (void)

**int** **LHSendEmail**()

{

usiEmailDummy = 0;

usiEmail1xTime2xIFTTTemail3 = 1; //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

\_i32 retVal = -1;

retVal = initializeAppVariables();

ASSERT\_ON\_ERROR(retVal);

// /\* Stop WDT and initialize the system-clock of the MCU \*/

// stopWDT();

// initClk();

/\* Configure command line interface \*/

CLI\_Configure();

displayBanner();

/\*

\* Following function configures the device to default state by cleaning

\* the persistent settings stored in NVMEM (viz. connection profiles &

\* policies, power policy etc)

\*

\* Applications may choose to skip this step if the developer is sure

\* that the device is in its default state at start of application

\*

\* Note that all profiles and persistent settings that were done on the

\* device will be lost

\*/

retVal = configureSimpleLinkToDefaultState();

**if**(retVal < 0)

{

**if** (*DEVICE\_NOT\_IN\_STATION\_MODE* == retVal)

CLI\_Write(" Failed to configure the device in its default state \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// LOOP\_FOREVER();

}

CLI\_Write(" Device is configured in default state \n\r");

/\*

\* Initializing the CC3100 device

\* Assumption is that the device is configured in station mode already

\* and it is in its default state

\*/

retVal = sl\_Start(0, 0, 0);

**if** ((retVal < 0) ||

(*ROLE\_STA* != retVal) )

{

CLI\_Write(" Failed to start the device \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// LOOP\_FOREVER();

}

CLI\_Write(" Device started as STATION \n\r");

/\* Configure the source email \*/

retVal = setEmail();

**if**(retVal < 0)

LOOP\_FOREVER();

/\* Connecting to WLAN AP \*/

retVal = establishConnectionWithAP();

**if**(retVal < 0)

{

CLI\_Write(" Failed to establish connection w/ an AP \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// LOOP\_FOREVER();

}

CLI\_Write(" Connection established w/ AP and IP is acquired \n\r");

/\* Configure and send the email \*/

CLI\_Write(" Sending email... \n\r");

retVal = sendEmail();

**if** (retVal < 0)

{

CLI\_Write(" Device couldn't send the email \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// LOOP\_FOREVER();

}

**else**

{

CLI\_Write(" Email Sent successfully \n\r");

}

/\* Disconnect from AP \*/

retVal = disconnectFromAP();

**if**(retVal < 0)

{

CLI\_Write(" Failed to disconnect from the AP \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// LOOP\_FOREVER();

}

retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// if(retVal < 0)

// LOOP\_FOREVER();

**return** 0;

} // Closing bracket for int LHSendEmail()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// End of function.....Start of function int LHSendEmail()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* ASYNCHRONOUS EVENT HANDLERS -- Start

\*/

/\*!

\brief This function handles WLAN events

\param[in] pWlanEvent is the event passed to the handler

\return None

\note

\warning

\*/

**void** **SimpleLinkWlanEventHandler**(SlWlanEvent\_t \*pWlanEvent)

{

**if**(pWlanEvent == NULL)

{

CLI\_Write(" [WLAN EVENT] NULL Pointer Error \n\r");

**return**;

}

**switch**(pWlanEvent->Event)

{

**case** SL\_WLAN\_CONNECT\_EVENT:

{

SET\_STATUS\_BIT(g\_Status, *STATUS\_BIT\_CONNECTION*);

/\*

\* Information about the connected AP (like name, MAC etc.) will be

\* available in 'slWlanConnectAsyncResponse\_t' - Applications

\* can use it if required

\*

\* slWlanConnectAsyncResponse\_t \*pEventData = NULL;

\* pEventData = &pWlanEvent->EventData.STAandP2PModeWlanConnected;

\*

\*/

}

**break**;

**case** SL\_WLAN\_DISCONNECT\_EVENT:

{

slWlanConnectAsyncResponse\_t\* pEventData = NULL;

CLR\_STATUS\_BIT(g\_Status, *STATUS\_BIT\_CONNECTION*);

CLR\_STATUS\_BIT(g\_Status, *STATUS\_BIT\_IP\_ACQUIRED*);

pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;

/\* If the user has initiated 'Disconnect' request, 'reason\_code' is SL\_USER\_INITIATED\_DISCONNECTION \*/

**if**(SL\_WLAN\_DISCONNECT\_USER\_INITIATED\_DISCONNECTION == pEventData->reason\_code)

{

CLI\_Write(" Device disconnected from the AP on application's request \n\r");

}

**else**

{

CLI\_Write(" Device disconnected from the AP on an ERROR..!! \n\r");

}

}

**break**;

**default**:

{

CLI\_Write(" [WLAN EVENT] Unexpected event \n\r");

}

**break**;

}

}

/\*!

\brief This function handles events for IP address acquisition via DHCP

indication

\param[in] pNetAppEvent is the event passed to the handler

\return None

\note

\warning

\*/

**void** **SimpleLinkNetAppEventHandler**(SlNetAppEvent\_t \*pNetAppEvent)

{

**if**(pNetAppEvent == NULL)

{

CLI\_Write(" [NETAPP EVENT] NULL Pointer Error \n\r");

**return**;

}

**switch**(pNetAppEvent->Event)

{

**case** SL\_NETAPP\_IPV4\_IPACQUIRED\_EVENT:

{

SET\_STATUS\_BIT(g\_Status, *STATUS\_BIT\_IP\_ACQUIRED*);

/\*

\* Information about the connected AP's IP, gateway, DNS etc.

\* will be available in 'SlIpV4AcquiredAsync\_t' - Applications

\* can use it if required

\*

\* SlIpV4AcquiredAsync\_t \*pEventData = NULL;

\* pEventData = &pNetAppEvent->EventData.ipAcquiredV4;

\* <gateway\_ip> = pEventData->gateway;

\*

\*/

}

**break**;

**default**:

{

CLI\_Write(" [NETAPP EVENT] Unexpected event \n\r");

}

**break**;

}

}

/\*!

\brief This function handles callback for the HTTP server events

\param[in] pHttpEvent - Contains the relevant event information

\param[in] pHttpResponse - Should be filled by the user with the

relevant response information

\return None

\note

\warning

\*/

**void** **SimpleLinkHttpServerCallback**(SlHttpServerEvent\_t \*pHttpEvent,

SlHttpServerResponse\_t \*pHttpResponse)

{

/\*

\* This application doesn't work with HTTP server - Hence these

\* events are not handled/expected here

\*/

CLI\_Write(" [HTTP EVENT] Unexpected event \n\r");

}

/\*!

\brief This function handles general error events indication

\param[in] pDevEvent is the event passed to the handler

\return None

\*/

**void** **SimpleLinkGeneralEventHandler**(SlDeviceEvent\_t \*pDevEvent)

{

/\*

\* Most of the general errors are not FATAL are are to be handled

\* appropriately by the application

\*/

CLI\_Write(" [GENERAL EVENT] \n\r");

}

/\*!

\brief This function handles socket events indication

\param[in] pSock is the event passed to the handler

\return None

\*/

**void** **SimpleLinkSockEventHandler**(SlSockEvent\_t \*pSock)

{

**if**(pSock == NULL)

{

CLI\_Write(" [SOCK EVENT] NULL Pointer Error \n\r");

**return**;

}

**switch**( pSock->Event )

{

**case** SL\_SOCKET\_TX\_FAILED\_EVENT:

{

/\*

\* TX Failed

\*

\* Information about the socket descriptor and status will be

\* available in 'SlSockEventData\_t' - Applications can use it if

\* required

\*

\* SlSockEventData\_u \*pEventData = NULL;

\* pEventData = & pSock->socketAsyncEvent;

\*/

**switch**( pSock->socketAsyncEvent.SockTxFailData.status )

{

**case** SL\_ECLOSE:

CLI\_Write(" [SOCK EVENT] Close socket operation failed to transmit all queued packets\n\r");

**break**;

**default**:

CLI\_Write(" [SOCK EVENT] Unexpected event \n\r");

**break**;

}

}

**break**;

**default**:

CLI\_Write(" [SOCK EVENT] Unexpected event \n\r");

**break**;

}

}

/\*

\* ASYNCHRONOUS EVENT HANDLERS -- End

\*/

/\*!

\brief This function configures the source email using parameters defined

in "config.h" file

\param[in] none

\return none

\note

\warning

\*/

**static** \_i32 **setEmail**()

{

\_i32 retVal = -1;

SlNetAppSourceEmail\_t sourceEmailId = {0};

SlNetAppSourcePassword\_t sourceEmailPwd = {0};

SlNetAppEmailOpt\_t eMailServerSetting = {0};

pal\_Memcpy(sourceEmailId.Username,USER,pal\_Strlen(USER)+1);

retVal = sl\_NetAppEmailSet(NETAPP\_SOURCE\_EMAIL,pal\_Strlen(USER)+1,

(\_u8\*)&sourceEmailId);

ASSERT\_ON\_ERROR(retVal);

pal\_Memcpy(sourceEmailPwd.Password,PASS,pal\_Strlen(PASS)+1);

retVal = sl\_NetAppEmailSet(NETAPP\_PASSWORD,pal\_Strlen(PASS)+1,

(\_u8\*)&sourceEmailPwd);

ASSERT\_ON\_ERROR(retVal);

eMailServerSetting.Family = AF\_INET;

eMailServerSetting.Port = GMAIL\_HOST\_PORT;

eMailServerSetting.Ip = SL\_IPV4\_VAL(74,125,129,108);

eMailServerSetting.SecurityMethod = SL\_SO\_SEC\_METHOD\_TLSV1\_2;

eMailServerSetting.SecurityCypher = SL\_SEC\_MASK\_TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA;

retVal = sl\_NetAppEmailSet(NETAPP\_ADVANCED\_OPT,**sizeof**(SlNetAppEmailOpt\_t),

(\_u8\*)&eMailServerSetting);

ASSERT\_ON\_ERROR(retVal);

**return** SUCCESS;

}

/\*!

\brief Send the email to the preconfigured email ID

\param[in] none

\return 0 for success , -1 otherwise

\note

\warning

\*/

**static** \_i32 **sendEmail**()

{

\_i32 Status = -1;

Status = sl\_NetAppEmailSet(NETAPP\_DEST\_EMAIL,

pal\_Strlen(DESTINATION\_EMAIL),

(\_u8 \*)DESTINATION\_EMAIL);

ASSERT\_ON\_ERROR(Status);

// Status = sl\_NetAppEmailSet(NETAPP\_SUBJECT,

// pal\_Strlen(EMAIL\_SUBJECT),

// (\_u8 \*)EMAIL\_SUBJECT);

//LH added 5-20-2020 to use my character array for the email subject

Status = sl\_NetAppEmailSet(NETAPP\_SUBJECT,

pal\_Strlen(LHemailSubject),

(\_u8 \*)LHemailSubject);

ASSERT\_ON\_ERROR(Status);

// Status = sl\_NetAppEmailSet(NETAPP\_MESSAGE,

// pal\_Strlen(EMAIL\_MESSAGE),

// (\_u8 \*)EMAIL\_MESSAGE);

//LH added 5-20-2020 to use my character array for the email message

Status = sl\_NetAppEmailSet(NETAPP\_MESSAGE,

pal\_Strlen(LHemailBody),

(\_u8 \*)LHemailBody);

ASSERT\_ON\_ERROR(Status);

Status = sl\_NetAppEmailConnect();

ASSERT\_ON\_ERROR(Status);

Status = sl\_NetAppEmailSend();

ASSERT\_ON\_ERROR(Status);

**return** SUCCESS;

}

/\*!

\brief This function configure the SimpleLink device in its default state. It:

- Sets the mode to STATION

- Configures connection policy to Auto and AutoSmartConfig

- Deletes all the stored profiles

- Enables DHCP

- Disables Scan policy

- Sets Tx power to maximum

- Sets power policy to normal

- Unregisters mDNS services

- Remove all filters

\param[in] none

\return On success, zero is returned. On error, negative is returned

\*/

**static** \_i32 **configureSimpleLinkToDefaultState**()

{

SlVersionFull ver = {0};

\_WlanRxFilterOperationCommandBuff\_t RxFilterIdMask = {0};

\_u8 val = 1;

\_u8 configOpt = 0;

\_u8 configLen = 0;

\_u8 power = 0;

\_i32 retVal = -1;

\_i32 mode = -1;

mode = sl\_Start(0, 0, 0);

ASSERT\_ON\_ERROR(mode);

/\* If the device is not in station-mode, try configuring it in station-mode \*/

**if** (*ROLE\_STA* != mode)

{

**if** (*ROLE\_AP* == mode)

{

/\* If the device is in AP mode, we need to wait for this event before doing anything \*/

**while**(!IS\_IP\_ACQUIRED(g\_Status)) { \_SlNonOsMainLoopTask(); }

}

/\* Switch to STA role and restart \*/

retVal = sl\_WlanSetMode(*ROLE\_STA*);

ASSERT\_ON\_ERROR(retVal);

retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

ASSERT\_ON\_ERROR(retVal);

retVal = sl\_Start(0, 0, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Check if the device is in station again \*/

**if** (*ROLE\_STA* != retVal)

{

/\* We don't want to proceed if the device is not coming up in station-mode \*/

ASSERT\_ON\_ERROR(*DEVICE\_NOT\_IN\_STATION\_MODE*);

}

}

/\* Get the device's version-information \*/

configOpt = SL\_DEVICE\_GENERAL\_VERSION;

configLen = **sizeof**(ver);

retVal = sl\_DevGet(SL\_DEVICE\_GENERAL\_CONFIGURATION, &configOpt, &configLen, (\_u8 \*)(&ver));

ASSERT\_ON\_ERROR(retVal);

/\* Set connection policy to Auto + SmartConfig (Device's default connection policy) \*/

retVal = sl\_WlanPolicySet(SL\_POLICY\_CONNECTION, SL\_CONNECTION\_POLICY(1, 0, 0, 0, 1), NULL, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Remove all profiles \*/

retVal = sl\_WlanProfileDel(0xFF);

ASSERT\_ON\_ERROR(retVal);

/\*

\* Device in station-mode. Disconnect previous connection if any

\* The function returns 0 if 'Disconnected done', negative number if already disconnected

\* Wait for 'disconnection' event if 0 is returned, Ignore other return-codes

\*/

retVal = sl\_WlanDisconnect();

**if**(0 == retVal)

{

/\* Wait \*/

**while**(IS\_CONNECTED(g\_Status)) { \_SlNonOsMainLoopTask(); }

}

/\* Enable DHCP client\*/

retVal = sl\_NetCfgSet(*SL\_IPV4\_STA\_P2P\_CL\_DHCP\_ENABLE*,1,1,&val);

ASSERT\_ON\_ERROR(retVal);

/\* Disable scan \*/

configOpt = SL\_SCAN\_POLICY(0);

retVal = sl\_WlanPolicySet(SL\_POLICY\_SCAN , configOpt, NULL, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Set Tx power level for station mode

Number between 0-15, as dB offset from max power - 0 will set maximum power \*/

power = 0;

retVal = sl\_WlanSet(SL\_WLAN\_CFG\_GENERAL\_PARAM\_ID, WLAN\_GENERAL\_PARAM\_OPT\_STA\_TX\_POWER, 1, (\_u8 \*)&power);

ASSERT\_ON\_ERROR(retVal);

/\* Set PM policy to normal \*/

retVal = sl\_WlanPolicySet(SL\_POLICY\_PM , SL\_NORMAL\_POLICY, NULL, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Unregister mDNS services \*/

retVal = sl\_NetAppMDNSUnRegisterService(0, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Remove all 64 filters (8\*8) \*/

pal\_Memset(RxFilterIdMask.FilterIdMask, 0xFF, 8);

retVal = sl\_WlanRxFilterSet(SL\_REMOVE\_RX\_FILTER, (\_u8 \*)&RxFilterIdMask,

**sizeof**(\_WlanRxFilterOperationCommandBuff\_t));

ASSERT\_ON\_ERROR(retVal);

retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

ASSERT\_ON\_ERROR(retVal);

retVal = initializeAppVariables();

ASSERT\_ON\_ERROR(retVal);

**return** retVal; /\* Success \*/

}

/\*!

\brief Connecting to a WLAN Access point

This function connects to the required AP (SSID\_NAME).

The function will return once we are connected and have acquired IP address

\param[in] None

\return 0 on success, negative error-code on error

\note

\warning If the WLAN connection fails or we don't acquire an IP address,

We will be stuck in this function forever.

\*/

**static** \_i32 **establishConnectionWithAP**()

{

**if** (usiEmail1xTime2xIFTTTemail3 == 1){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

SlSecParams\_t secParams = {0};

\_i32 retVal = 0;

secParams.Key = PASSKEY;

secParams.KeyLen = PASSKEY\_LEN;

secParams.Type = SEC\_TYPE;

retVal = sl\_WlanConnect(SSID\_NAME, pal\_Strlen(SSID\_NAME), 0, &secParams, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Wait \*/

**while**((!IS\_CONNECTED(g\_Status)) || (!IS\_IP\_ACQUIRED(g\_Status))) { \_SlNonOsMainLoopTask(); }

// return SUCCESS;

}

**else** **if** (usiEmail1xTime2xIFTTTemail3 == 2){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

SlSecParams\_t secParams = {0};

\_i32 retVal = 0;

secParams.Key = PASSKEY;

secParams.KeyLen = pal\_Strlen(secParams.Key);

secParams.Type = SEC\_TYPE;

retVal = sl\_WlanConnect(SSID\_NAME, pal\_Strlen(SSID\_NAME), 0, &secParams, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Wait \*/

**while**((!IS\_CONNECTED(g\_Status)) || (!IS\_IP\_ACQUIRED(g\_Status))) { \_SlNonOsMainLoopTask(); }

// return SUCCESS;

}

**else** **if** (usiEmail1xTime2xIFTTTemail3 == 3){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

SlSecParams\_t secParams = {0};

\_i32 retVal = 0;

secParams.Key = PASSKEY;

secParams.KeyLen = pal\_Strlen(PASSKEY);

secParams.Type = SEC\_TYPE;

retVal = sl\_WlanConnect(SSID\_NAME, pal\_Strlen(SSID\_NAME), 0, &secParams, 0);

ASSERT\_ON\_ERROR(retVal);

/\* Wait \*/

**while**((!IS\_CONNECTED(g\_Status)) || (!IS\_IP\_ACQUIRED(g\_Status))) { \_SlNonOsMainLoopTask(); }

// return SUCCESS;

}

**return** SUCCESS;

}

/\*!

\brief Disconnecting from a WLAN Access point

This function disconnects from the connected AP

\param[in] None

\return none

\note

\warning If the WLAN disconnection fails, we will be stuck in this function forever.

\*/

**static** \_i32 **disconnectFromAP**()

{

\_i32 retVal = -1;

/\*

\* The function returns 0 if 'Disconnected done', negative number if already disconnected

\* Wait for 'disconnection' event if 0 is returned, Ignore other return-codes

\*/

retVal = sl\_WlanDisconnect();

**if**(0 == retVal)

{

/\* Wait \*/

**while**(IS\_CONNECTED(g\_Status)) { \_SlNonOsMainLoopTask(); }

}

**return** SUCCESS;

}

/\*!

\brief This function initializes the application variables

\param[in] None

\return 0 on success, negative error-code on error

\*/

**static** \_i32 **initializeAppVariables**()

{

**if** (usiEmail1xTime2xIFTTTemail3 == 1){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

g\_Status = 0;

// return SUCCESS;

}

**else** **if** (usiEmail1xTime2xIFTTTemail3 == 2){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

g\_Status = 0;

pal\_Memset(&appData, 0, **sizeof**(appData)); //This line not in email wifi call

// return SUCCESS;

}

**else** **if** (usiEmail1xTime2xIFTTTemail3 == 3){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

g\_Status = 0;

g\_SockID = 0;

g\_DestinationIP = 0;

g\_BytesReceived = 0;

pal\_Memset(g\_buff, 0, **sizeof**(g\_buff));

// return SUCCESS;

}

**return** SUCCESS;

}

/\*!

\brief This function displays the application's banner

\param None

\return None

\*/

**static** **void** **displayBanner**()

{

usiEmailDummy = 0;

**if** (usiEmail1xTime2xIFTTTemail3 == 1){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

CLI\_Write("\n\r\n\r");

CLI\_Write(" Email application - Version ");

CLI\_Write(APPLICATION\_VERSION);

CLI\_Write("\n\r\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");

}

**else** **if** (usiEmail1xTime2xIFTTTemail3 == 2){ //In the function call set to 1 for email, set to 2 for get-time

usiEmailDummy = 0;

CLI\_Write((\_u8 \*)"\n\r\n\r"); //LH Line 0 = blank

CLI\_Write((\_u8 \*)" Get time application - Version "); //LH line 1

CLI\_Write(APPLICATION\_VERSION); //LH last half of line 1

CLI\_Write((\_u8 \*)"\n\r\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r"); //LH Line 2 \*\*\* under top line

}

**else** **if** (usiEmail1xTime2xIFTTTemail3 == 3){ //In the function call set to 1 for email, set to 2 for get-time, 3 for IFTTT email

CLI\_Write("\n\r\n\r");

CLI\_Write(" HTTP Client - Version ");

CLI\_Write(APPLICATION\_VERSION);

CLI\_Write("\n\r\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH

//\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH

//\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH\*\*\*\*get-timeLH

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* get-time-LH.c

\*

\* Created on: Jul 30, 2020

\* Author: L Hinz

\*/

/\*

\* Application Name - Get time

\* Application Overview - This sample application demonstrates how

\* to connect to an SNTP server and request for time

\* information. The application processes the received

\* data and displays the time on console

\* Application Details - http://processors.wiki.ti.com/index.php/CC31xx\_Get\_Time\_Application

\* doc\examples\get\_time.pdf

\*/

//#include "simplelink.h" //8-1-2020 comment out as duplicate with email file above

//#include "sl\_common.h" //8-1-2020 comment out as duplicate with email file above

//#include "email.h"

//#include "get-time-LH.h"

//#include "base64.h"

//#include "config.h"

//#include "sl\_common.h"

**#include** <stdio.h> //For printf function

**#define** APPLICATION\_VERSION "1.3.0"

**#define** SL\_STOP\_TIMEOUT 0xFF

**#define** MAX\_BUF\_SIZE 48

**#define** TIME2013 3566160000 /\* 43 years + 13 days of leap years \*/ //LH try adding 2 more days of leap years seconds

//This gives the correct date number but as expected the Day of the week is then 2 days advanced in error

//I do not use the Day of the week at this time, so OK

//#define TIME2013 3565987200 /\* 43 years + 11 days of leap years \*/ //LH 7-29-2020; Date # is 2 days advanced

**#define** YEAR2013 2013

**#define** SEC\_IN\_MIN 60

**#define** SEC\_IN\_HOUR 3600

**#define** SEC\_IN\_DAY 86400

/\*

\* Values for below macros shall be modified for setting the time-zone

\*/

**#define** GMT\_TIME\_ZONE\_HR -5 //\*\*\*\*Note that our time zone needs to be entered as a negative number

**#define** GMT\_TIME\_ZONE\_MIN 00

/\* Application specific status/error codes \*/

//LH 8-2-2020 added these to config.h line 70 otherwise this typedef define twice

//typedef enum{

// DEVICE\_NOT\_IN\_STATION\_MODE = -0x7D0, /\* Choosing this number to avoid overlap w/ host-driver's error codes \*/

// SNTP\_SEND\_ERROR = DEVICE\_NOT\_IN\_STATION\_MODE - 1,

// SNTP\_RECV\_ERROR = SNTP\_SEND\_ERROR - 1,

// SNTP\_SERVER\_RESPONSE\_ERROR = SNTP\_RECV\_ERROR - 1,

//

// STATUS\_CODE\_MAX = -0xBB8

//}e\_AppStatusCodes;

/\*

\* GLOBAL VARIABLES -- Start

\*/

//\_u32 g\_Status = 0; //8-1-2020 comment out as duplicate with email file above

//LH 8-2-2020 getting errors that appData is not defined so try pasting it in email.h, I added the word typedef in front to make it the same

// Doing that generates 100's of errors, every place appData was used; took word typdef off and get a clean compile????

//struct{

// \_u32 DestinationIP;

// \_u32 elapsedSec;

// \_u32 uGeneralVar;

// \_u32 uGeneralVar1;

//

// \_u16 ccLen;

//

// \_i32 SockID;

// \_i32 sGeneralVar;

//

// \_u8 time[30]; //LH time is a structure with 30 elements so it can display a string like this Tue Oct 10 2018 11:26:13

// \_u8 \*ccPtr;

//

//}appData;

**volatile** **unsigned** **short** usiBreakDummy = 0;

**volatile** **unsigned** **short** usiYearGetTime = 0;

**volatile** **unsigned** **short** usiMonthOfYearGetTime = 0;

**volatile** **unsigned** **short** usiDayOfMonthGetTime = 0;

**volatile** **unsigned** **short** usiDayOfWeekGetTime = 0;

**volatile** **unsigned** **short** usiHrsGetTime = 0;

**volatile** **unsigned** **short** usiMinutesGetTime = 0;

**volatile** **unsigned** **short** usiSecondsGetTime = 0;

**char** ChrHrTens = '\0'; //Initialized to null

/\*

\* GLOBAL VARIABLES -- End

\*/

/\*

\* CONTSTANT -- Start

\*/

/\*! ######################### List of SNTP servers ############################

\*! ##

\*! ## hostname | IP | location

\*! ## ------------------------------------------------------------------------

\*! ## time-a.nist.gov | 129.6.15.28 | NIST, Gaithersburg, Maryland

\*! ## time-b.nist.gov | 129.6.15.29 | NIST, Gaithersburg, Maryland

\*! ## time-c.nist.gov | 129.6.15.30 | NIST, Gaithersburg, Maryland

\*! ## nist1-macon.macon.ga.us | 98.175.203.200 | Macon, Georgia

\*! ## 0.in.pool.ntp.org | 123.108.225.6 | India

\*! ## For more SNTP server link visit 'http://tf.nist.gov/tf-cgi/servers.cgi'

\*! ###########################################################################

\*/

//const \_u8 SNTPserver[30] = "time-a.nist.gov"; /\* Add any one of the above servers \*/

//const \_u8 SNTPserver[30] = "time-b.nist.gov"; /\* Add any one of the above servers \*/

**const** \_u8 SNTPserver[30] = "time-c.nist.gov"; /\* Add any one of the above servers \*/

//const \_u8 SNTPserver[30] = "nist1-macon.macon.ga.us"; /\* Add any one of the above servers \*/ //LH-7-29-2020 never got a reply

/\* Tuesday is the 1st day in 2013 - the relative year \*/

**const** \_u8 daysOfWeek2013[7][3] = {{"Tue"},

{"Wed"},

{"Thu"},

{"Fri"},

{"Sat"},

{"Sun"},

{"Mon"}};

**const** \_u8 monthOfYear[12][3] = {{"Jan"},

{"Feb"},

{"Mar"},

{"Apr"},

{"May"},

{"Jun"},

{"Jul"},

{"Aug"},

{"Sep"},

{"Oct"},

{"Nov"},

{"Dec"}};

**const** \_u8 numOfDaysPerMonth[12] = {31,28,31,30,31,30,31,31,30,31,30,31};

**const** \_u8 digits[] = "0123456789";

/\*

\* CONTSTANT -- End

\*/

/\*

\* STATIC FUNCTION DEFINITIONS -- Start

\*/

//static \_i32 establishConnectionWithAP(); //Already defined at around line 700

//static \_i32 disconnectFromAP(); //Already defined at line 792

//static \_i32 configureSimpleLinkToDefaultState();

//static \_i32 initializeAppVariables(); //Already defined at around line 795

//static void displayBanner(); //Already defined at line 796

//LH even in the orginal code these functions are not in the get-time file, must be in sl-common or some other WiFi library

**static** \_i32 **getHostIP**();

**static** \_i32 **createConnection**();

**static** \_i32 **getSNTPTime**(\_i16 gmt\_hr, \_i16 gmt\_min);

**static** \_u16 **itoa**(\_i16 cNum, \_u8 \*cString);

/\*

\* STATIC FUNCTION DEFINITIONS -- End

\*/

/\*

\* Application's entry point

\*/

**int** **LHGetTime** ()

//int main(int argc, char\*\* argv)

{

usiEmail1xTime2xIFTTTemail3 = 2; //In the function call set to 1 for email, set to 2 for get-time

//// 4-27-2023 LH add sl\_Stop

// usiBreakDummy = 0;

//// sl\_Stop(SL\_STOP\_TIMEOUT); //Never times out

// sl\_Stop(5);

// usiBreakDummy = 0;

\_i32 retVal = -1;

retVal = initializeAppVariables();

ASSERT\_ON\_ERROR(retVal);

/\* Stop WDT and initialize the system-clock of the MCU \*/

// stopWDT();

// initClk();

/\* Configure command line interface \*/

CLI\_Configure();

displayBanner();

/\*

\* Following function configures the device to default state by cleaning

\* the persistent settings stored in NVMEM (viz. connection profiles &

\* policies, power policy etc)

\*

\* Applications may choose to skip this step if the developer is sure

\* that the device is in its default state at start of application

\*

\* Note that all profiles and persistent settings that were done on the

\* device will be lost

\*/

retVal = configureSimpleLinkToDefaultState();

**if**(retVal < 0)

{

**if** (*DEVICE\_NOT\_IN\_STATION\_MODE* == retVal)

CLI\_Write((\_u8 \*)" Failed to configure the device in its default state \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

//LOOP\_FOREVER();

}

CLI\_Write((\_u8 \*)" Device is configured in default state \n\r"); //LH line 3

/\*

\* Initializing the CC3100 device

\* Assumption is that the device is configured in station mode already

\* and it is in its default state

\*/

retVal = sl\_Start(0, 0, 0);

**if** ((retVal < 0) ||

(*ROLE\_STA* != retVal) )

{

CLI\_Write((\_u8 \*)" Failed to start the device \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

//LOOP\_FOREVER();

}

CLI\_Write((\_u8 \*)" Device started as STATION \n\r"); //LH line 4

/\* Connecting to WLAN AP \*/

retVal = establishConnectionWithAP();

**if**(retVal < 0)

{

CLI\_Write((\_u8 \*)" Failed to establish connection w/ an AP \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

//LOOP\_FOREVER();

}

CLI\_Write(" Connection established w/ AP and IP is acquired \n\r"); //LH Line 5

//8-2-2020 8:30 PM Does complete the above statement, then no further

retVal = getHostIP();

CLI\_Write(" LH Just after get HostIP \n\r");

**if**(retVal < 0)

{

CLI\_Write((\_u8 \*)" Unable to get host IP\n\r\n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

//LOOP\_FOREVER();

}

appData.SockID = createConnection();

CLI\_Write(" LH Just after createConnection \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// if(appData.SockID < 0)

//LOOP\_FOREVER();

CLI\_Write(" LH Just BEFORE getSNTPTTime... \n\r");

retVal = getSNTPTime(GMT\_TIME\_ZONE\_HR,GMT\_TIME\_ZONE\_MIN);

CLI\_Write(" LH Just AFTER getSNTPTTime... \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// if(retVal < 0)

//LOOP\_FOREVER();

usiBreakDummy = 0;

retVal = sl\_Close(appData.SockID);

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

// if(retVal < 0)

//LOOP\_FOREVER();

usiBreakDummy = 0;

retVal = disconnectFromAP();

**if**(retVal < 0)

{

CLI\_Write((\_u8 \*)" Failed to disconnect from the AP \n\r");

// 5-24-2020, I don't see the value of LOOP\_FOREVER(); as this just hangs the program forever at this point

//I think better to move on, as already printed out the failure, maybe some time later it will work

//LOOP\_FOREVER();

}

//LH added 7-31-2020

usiBreakDummy = 0;

retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

**return** 0;

}

//\*\*\*\*\*\*End of Application Function called from main\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* ASYNCHRONOUS EVENT HANDLERS -- Start

\*/

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function handles WLAN events

//

// \param[in] pWlanEvent is the event passed to the handler

//

// \return None

//

// \note

//

// \warning

//\*/

//void SimpleLinkWlanEventHandler(SlWlanEvent\_t \*pWlanEvent)

//{

// if(pWlanEvent == NULL)

// {

// CLI\_Write(" [WLAN EVENT] NULL Pointer Error \n\r");

// return;

// }

//

// switch(pWlanEvent->Event)

// {

// case SL\_WLAN\_CONNECT\_EVENT:

// {

// SET\_STATUS\_BIT(g\_Status, STATUS\_BIT\_CONNECTION);

//

// /\*

// \* Information about the connected AP (like name, MAC etc) will be

// \* available in 'slWlanConnectAsyncResponse\_t' - Applications

// \* can use it if required

// \*

// \* slWlanConnectAsyncResponse\_t \*pEventData = NULL;

// \* pEventData = &pWlanEvent->EventData.STAandP2PModeWlanConnected;

// \*

// \*/

// }

// break;

//

// case SL\_WLAN\_DISCONNECT\_EVENT:

// {

// slWlanConnectAsyncResponse\_t\* pEventData = NULL;

//

// CLR\_STATUS\_BIT(g\_Status, STATUS\_BIT\_CONNECTION);

// CLR\_STATUS\_BIT(g\_Status, STATUS\_BIT\_IP\_ACQUIRED);

//

// pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;

//

// /\* If the user has initiated 'Disconnect' request, 'reason\_code' is SL\_USER\_INITIATED\_DISCONNECTION \*/

// if(SL\_WLAN\_DISCONNECT\_USER\_INITIATED\_DISCONNECTION == pEventData->reason\_code)

// {

// CLI\_Write((\_u8 \*)" Device disconnected from the AP on application's request \n\r"); //LH Line 11, last line

// }

// else

// {

// CLI\_Write((\_u8 \*)" Device disconnected from the AP on an ERROR..!! \n\r");

// }

// }

// break;

//

// default:

// {

// CLI\_Write((\_u8 \*)" [WLAN EVENT] Unexpected event \n\r");

// }

// break;

// }

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function handles events for IP address acquisition via DHCP

// indication

//

// \param[in] pNetAppEvent is the event passed to the handler

//

// \return None

//

// \note

//

// \warning

//\*/

//void SimpleLinkNetAppEventHandler(SlNetAppEvent\_t \*pNetAppEvent)

//{

// if(pNetAppEvent == NULL)

// {

// CLI\_Write(" [NETAPP EVENT] NULL Pointer Error \n\r");

// return;

// }

//

// switch(pNetAppEvent->Event)

// {

// case SL\_NETAPP\_IPV4\_IPACQUIRED\_EVENT:

// {

// SET\_STATUS\_BIT(g\_Status, STATUS\_BIT\_IP\_ACQUIRED);

//

// /\*

// \* Information about the connected AP's IP, gateway, DNS etc

// \* will be available in 'SlIpV4AcquiredAsync\_t' - Applications

// \* can use it if required

// \*

// \* SlIpV4AcquiredAsync\_t \*pEventData = NULL;

// \* pEventData = &pNetAppEvent->EventData.ipAcquiredV4;

// \* <gateway\_ip> = pEventData->gateway;

// \*

// \*/

// }

// break;

//

// default:

// {

// CLI\_Write((\_u8 \*)" [NETAPP EVENT] Unexpected event \n\r");

// }

// break;

// }

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function handles callback for the HTTP server events

//

// \param[in] pHttpEvent - Contains the relevant event information

// \param[in] pHttpResponse - Should be filled by the user with the

// relevant response information

//

// \return None

//

// \note

//

// \warning

//\*/

//void SimpleLinkHttpServerCallback(SlHttpServerEvent\_t \*pHttpEvent,

// SlHttpServerResponse\_t \*pHttpResponse)

//{

// /\*

// \* This application doesn't work with HTTP server - Hence these

// \* events are not handled here

// \*/

// CLI\_Write((\_u8 \*)" [HTTP EVENT] Unexpected event \n\r");

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function handles general error events indication

//

// \param[in] pDevEvent is the event passed to the handler

//

// \return None

//\*/

//void SimpleLinkGeneralEventHandler(SlDeviceEvent\_t \*pDevEvent)

//{

// /\*

// \* Most of the general errors are not FATAL are to be handled

// \* appropriately by the application

// \*/

// CLI\_Write((\_u8 \*)" [GENERAL EVENT] \n\r");

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function handles socket events indication

//

// \param[in] pSock is the event passed to the handler

//

// \return None

//\*/

//void SimpleLinkSockEventHandler(SlSockEvent\_t \*pSock)

//{

// if(pSock == NULL)

// {

// CLI\_Write(" [SOCK EVENT] NULL Pointer Error \n\r");

// return;

// }

//

// switch( pSock->Event )

// {

// case SL\_SOCKET\_TX\_FAILED\_EVENT:

// {

// /\*

// \* TX Failed

// \*

// \* Information about the socket descriptor and status will be

// \* available in 'SlSockEventData\_t' - Applications can use it if

// \* required

// \*

// \* SlSockEventData\_u \*pEventData = NULL;

// \* pEventData = & pSock->socketAsyncEvent;

// \*/

// switch( pSock->socketAsyncEvent.SockTxFailData.status )

// {

// case SL\_ECLOSE:

// CLI\_Write((\_u8 \*)" [SOCK EVENT] Close socket operation failed to transmit all queued packets\n\r");

// break;

//

//

// default:

// CLI\_Write((\_u8 \*)" [SOCK EVENT] Unexpected event \n\r");

// break;

// }

// }

// break;

//

// default:

// CLI\_Write((\_u8 \*)" [SOCK EVENT] Unexpected event \n\r");

// break;

// }

//}

/\*

\* ASYNCHRONOUS EVENT HANDLERS -- End

\*/

/\*!

\brief Convert integer to ASCII in decimal base

\param[in] cNum - integer number to convert

\param[OUT] cString - output string

\return number of ASCII characters

\warning

\*/

**static** \_u16 **itoa**(\_i16 cNum, \_u8 \*cString)

{

\_u16 length = 0;

\_u8\* ptr = NULL;

\_i16 uTemp = cNum;

/\* value 0 is a special case \*/

**if** (cNum == 0)

{

length = 1;

\*cString = '0';

**return** length;

}

/\* Find out the length of the number, in decimal base \*/

length = 0;

**while** (uTemp > 0)

{

uTemp /= 10;

length++;

}

/\* Do the actual formatting, right to left \*/

uTemp = cNum;

ptr = cString + length;

**while** (uTemp > 0)

{

--ptr;

\*ptr = digits[uTemp % 10];

uTemp /= 10;

}

**return** length;

}

/\*!

\brief Get the required data from the server.

\param[in] gmt\_hr - GMT offset hours

\param[in] gmt\_min - GMT offset minutes

\return 0 on success, -ve otherwise

\warning

\*/

**static** \_i32 **getSNTPTime**(\_i16 gmt\_hr, \_i16 gmt\_min)

{

/\*

NTP Packet Header:

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|LI | VN |Mode | Stratum | Poll | Precision |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Root Delay |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Root Dispersion |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Reference Identifier |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |

| Reference Time-stamp (64) |

| |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |

| Originate Time-stamp (64) |

| |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |

| Receive Time-stamp (64) |

| |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |

| Transmit Time-stamp (64) |

| |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Key Identifier (optional) (32) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| |

| |

| Message Digest (optional) (128) |

| |

| |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

\*/

SlSockAddrIn\_t LocalAddr;

SlSockAddr\_t Addr;

\_u8 dataBuf[MAX\_BUF\_SIZE];

\_i32 retVal = -1;

\_i16 AddrSize = 0;

/\* For time zone with negative GMT value, change minutes to negative for

\* computation \*/

**if**(gmt\_hr < 0 && gmt\_min > 0)

gmt\_min = gmt\_min \* (-1);

sl\_Memset(dataBuf, 0, **sizeof**(dataBuf));

dataBuf[0] = '\x1b';

Addr.sa\_family = AF\_INET;

/\* the source port \*/

Addr.sa\_data[0] = 0x00;

Addr.sa\_data[1] = 0x7B; /\* 123 \*/

Addr.sa\_data[2] = (\_u8)((appData.DestinationIP >> 24) & 0xff);

Addr.sa\_data[3] = (\_u8)((appData.DestinationIP >> 16) & 0xff);

Addr.sa\_data[4] = (\_u8)((appData.DestinationIP >> 8) & 0xff);

Addr.sa\_data[5] = (\_u8) (appData.DestinationIP & 0xff);

retVal = sl\_SendTo(appData.SockID, dataBuf, **sizeof**(dataBuf), 0,

&Addr, **sizeof**(Addr));

**if** (retVal != **sizeof**(dataBuf))

{

/\* could not send SNTP request \*/

CLI\_Write((\_u8 \*)" Device couldn't send SNTP request\n\r\n\r");

ASSERT\_ON\_ERROR(*SNTP\_SEND\_ERROR*);

}

AddrSize = **sizeof**(SlSockAddrIn\_t);

LocalAddr.sin\_family = SL\_AF\_INET;

LocalAddr.sin\_port = 0;

LocalAddr.sin\_addr.s\_addr = 0;

retVal = sl\_Bind(appData.SockID,(SlSockAddr\_t \*)&LocalAddr, AddrSize);

**if**(retVal < 0)

ASSERT\_ON\_ERROR(retVal);

retVal = sl\_RecvFrom(appData.SockID, dataBuf, **sizeof**(dataBuf), 0,

(SlSockAddr\_t \*)&LocalAddr, (SlSocklen\_t\*)&AddrSize);

**if** (retVal <= 0)

{

CLI\_Write((\_u8 \*)" Device couldn't receive time information \n\r");

ASSERT\_ON\_ERROR(*SNTP\_RECV\_ERROR*);

}

**if** ((dataBuf[0] & 0x7) != 4) /\* expect only server response \*/

{

/\* MODE is not server, abort \*/

CLI\_Write((\_u8 \*)" Device is expecting response from server only!\n\r");

ASSERT\_ON\_ERROR(*SNTP\_SERVER\_RESPONSE\_ERROR*);

}

**else**

{

\_u8 index;

appData.elapsedSec = dataBuf[40];

appData.elapsedSec <<= 8;

appData.elapsedSec += dataBuf[41];

appData.elapsedSec <<= 8;

appData.elapsedSec += dataBuf[42];

appData.elapsedSec <<= 8;

appData.elapsedSec += dataBuf[43];

appData.elapsedSec -= TIME2013;

/\* correct the time zone \*/

appData.elapsedSec += (gmt\_hr \* SEC\_IN\_HOUR);

appData.elapsedSec += (gmt\_min \* SEC\_IN\_MIN);

appData.ccPtr = &appData.time[0];

/\* day \*/

// appData.sGeneralVar = appData.elapsedSec/SEC\_IN\_DAY;

appData.sGeneralVar = (appData.elapsedSec + 172800) /SEC\_IN\_DAY; //LH Day of week 2 days behind due to leap years

//This fixed the Day of week, but then re-broke the Date number

pal\_Memcpy(appData.ccPtr, daysOfWeek2013[appData.sGeneralVar%7], 3);

//Day of the week

usiDayOfWeekGetTime = appData.sGeneralVar%7;

/\* Tuesday is the 1st day in 2013 - the relative year \*/

//const \_u8 daysOfWeek2013[7][3] = {{"Tue"}, // 0 = Tuesday

// {"Wed"}, // 1 = Wednesday

// {"Thu"}, // 2 = Thursday

// {"Fri"}, // 3 = Friday

// {"Sat"}, // 4 = Saturday

// {"Sun"}, // 5 = Sunday

// {"Mon"}}; // 6 = Monday

appData.sGeneralVar = appData.elapsedSec /SEC\_IN\_DAY; //LH restore the sGeneralVar so date calculates correctly

appData.ccPtr += 3;

\*appData.ccPtr++ = '\x20';

/\* month \*/

appData.sGeneralVar %= 365;

**for** (index = 0; index < 12; index++)

{

appData.sGeneralVar -= numOfDaysPerMonth[index];

**if** (appData.sGeneralVar < 0)

**break**;

}

pal\_Memcpy(appData.ccPtr, monthOfYear[index], 3);

//Month of the Year

usiMonthOfYearGetTime = index + 1; //See Reference table at top of Get time section, Array starts at zero, so if want normal mth # add 1

appData.ccPtr += 3;

\*appData.ccPtr++ = '\x20';

/\* date \*/

/\* restore the day in current month\*/

appData.sGeneralVar += numOfDaysPerMonth[index];

appData.ccLen = itoa(appData.sGeneralVar + 1, appData.ccPtr); //Calls function itoa integer to ASCII decimal

//LH Convert integer to ASCII in decimal base; param[in] cNum - integer number to convert

//param[OUT] cString - output string; \return number of ASCII characters

//So at this point uGeneralVar must be a 2 digit integer that contains the day of the month

//Day of the Month

usiDayOfMonthGetTime = appData.sGeneralVar +1; //note this is s not u GeneralVar; Had to add the +1 like above, not sure why but do

usiBreakDummy = 0;

appData.ccPtr += appData.ccLen;

\*appData.ccPtr++ = '\x20';

/\* year \*/

/\* number of days since beginning of 2013 \*/

appData.uGeneralVar = appData.elapsedSec/SEC\_IN\_DAY;

appData.uGeneralVar /= 365;

appData.ccLen = itoa(YEAR2013 + appData.uGeneralVar , appData.ccPtr); //So this would be 2013 + 7 = 2020, so uGeneralVar must = # incr yr after 2013

appData.ccPtr += appData.ccLen;

\*appData.ccPtr++ = '\x20';

//Year

usiYearGetTime = YEAR2013 + appData.uGeneralVar;

/\* time \*/

appData.uGeneralVar = appData.elapsedSec%SEC\_IN\_DAY;

/\* number of seconds per hour \*/

appData.uGeneralVar1 = appData.uGeneralVar%SEC\_IN\_HOUR;

appData.uGeneralVar /= SEC\_IN\_HOUR; /\* number of hours \*/

appData.ccLen = itoa(appData.uGeneralVar, appData.ccPtr); //LH believe Hrs to Char

//So at this point uGeneralVar must be a 2 digit integer that contains hours

usiBreakDummy = 0;

//Hours

usiHrsGetTime = appData.uGeneralVar;

usiBreakDummy = 0;

appData.ccPtr += appData.ccLen;

\*appData.ccPtr++ = ':'; //LH believe colon between Hr and Min

/\* number of minutes per hour \*/

appData.uGeneralVar = appData.uGeneralVar1/SEC\_IN\_MIN;

/\* number of seconds per minute \*/

appData.uGeneralVar1 %= SEC\_IN\_MIN;

appData.ccLen = itoa(appData.uGeneralVar, appData.ccPtr); //LH believe minutes to Char

//So at this point uGeneralVar must be a 2 digit integer that contains minutes

//Minutes

usiMinutesGetTime = appData.uGeneralVar;

usiBreakDummy = 0;

appData.ccPtr += appData.ccLen;

\*appData.ccPtr++ = ':'; //LH believe colon between Min and Sec

appData.ccLen = itoa(appData.uGeneralVar1, appData.ccPtr); //LH believe seconds to Char

//So at this point uGeneralVar1 must be a 2 digit integer that contains seconds

//Seconds

usiSecondsGetTime = appData.uGeneralVar1;

usiBreakDummy = 0;

appData.ccPtr += appData.ccLen;

\*appData.ccPtr++ = '\x20';

\*appData.ccPtr++ = '\0';

CLI\_Write((\_u8 \*)"\r\n Server "); //LH Line 7

CLI\_Write((\_u8 \*)SNTPserver);

CLI\_Write((\_u8 \*)" has responded with time information");

CLI\_Write((\_u8 \*)"\n\r\r\n "); //LH Line 8 which is blank

CLI\_Write((\_u8 \*)appData.time); //LH when look at time structure it only uses 23 of the 30 elements in the structure; Line 9 with the time

CLI\_Write((\_u8 \*)"\n\r\r\n"); //LH line 10 which is a blank line

//time element 16 HrTens, element 17 HrUnits; element 19 MinTens, element 20 MinUnits, element 22 SecTens, element 23 SecUnits

//LH added 7-31-2020

//CLI is set up for characters, so it prints the characters but not the integers

// CLI\_Write((\_u8 \*)"\r\n usiDayOfMonth: ");

// CLI\_Write((\_u8 \*)usiDayOfMonth);

// CLI\_Write((\_u8 \*)"\r\n usiHrs: ");

// CLI\_Write((\_u8 \*)usiHrs);

// CLI\_Write((\_u8 \*)"\r\n usiMinutes: ");

// CLI\_Write((\_u8 \*)usiMinutes);

// CLI\_Write((\_u8 \*)"\r\n usiDayOfMonth: ");

// CLI\_Write((\_u8 \*)usiSeconds);

//printf compiles and runs but no seen on the console, not sure were it has gone to

// printf("\r\n usiDayOfMonth: %d", usiDayOfMonth);

// printf("\r\n usiHrs: %d", usiHrs);

// printf("\r\n usiMinutes: %d", usiMinutes);

// printf("\r\n usiSeconds: %d", usiSeconds);

}

**return** SUCCESS;

}

/\*!

\brief Create UDP socket to communicate with server.

\param[in] none

\return Socket descriptor for success otherwise negative

\warning

\*/

**static** \_i32 **createConnection**()

{

\_i32 sd = 0;

sd = sl\_Socket(SL\_AF\_INET, SL\_SOCK\_DGRAM, IPPROTO\_UDP);

**if**( sd < 0 )

{

CLI\_Write((\_u8 \*)"Error creating socket\n\r\n\r");

}

**return** sd;

}

/\*!

\brief Gets the Server IP address

\param[in] none

\return zero for success and -1 for error

\warning

\*/

**static** \_i32 **getHostIP**()

{

\_i32 status = 0;

appData.DestinationIP = 0;

status = sl\_NetAppDnsGetHostByName((\_i8\*)SNTPserver, pal\_Strlen(SNTPserver),

&appData.DestinationIP, SL\_AF\_INET);

ASSERT\_ON\_ERROR(status);

**return** SUCCESS;

}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function configure the SimpleLink device in its default state. It:

// - Sets the mode to STATION

// - Configures connection policy to Auto and AutoSmartConfig

// - Deletes all the stored profiles

// - Enables DHCP

// - Disables Scan policy

// - Sets Tx power to maximum

// - Sets power policy to normal

// - Unregisters mDNS services

// - Remove all filters

//

// \param[in] none

//

// \return On success, zero is returned. On error, negative is returned

//\*/

//static \_i32 configureSimpleLinkToDefaultState()

//{

// SlVersionFull ver = {0};

// \_WlanRxFilterOperationCommandBuff\_t RxFilterIdMask = {0};

//

// \_u8 val = 1;

// \_u8 configOpt = 0;

// \_u8 configLen = 0;

// \_u8 power = 0;

//

// \_i32 retVal = -1;

// \_i32 mode = -1;

//

// mode = sl\_Start(0, 0, 0);

// ASSERT\_ON\_ERROR(mode);

//

// /\* If the device is not in station-mode, try configuring it in station-mode \*/

// if (ROLE\_STA != mode)

// {

// if (ROLE\_AP == mode)

// {

// /\* If the device is in AP mode, we need to wait for this event before doing anything \*/

// while(!IS\_IP\_ACQUIRED(g\_Status)) { \_SlNonOsMainLoopTask(); }

// }

//

// /\* Switch to STA role and restart \*/

// retVal = sl\_WlanSetMode(ROLE\_STA);

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = sl\_Start(0, 0, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Check if the device is in station again \*/

// if (ROLE\_STA != retVal)

// {

// /\* We don't want to proceed if the device is not coming up in station-mode \*/

// ASSERT\_ON\_ERROR(DEVICE\_NOT\_IN\_STATION\_MODE);

// }

// }

//

// /\* Get the device's version-information \*/

// configOpt = SL\_DEVICE\_GENERAL\_VERSION;

// configLen = sizeof(ver);

// retVal = sl\_DevGet(SL\_DEVICE\_GENERAL\_CONFIGURATION, &configOpt, &configLen, (\_u8 \*)(&ver));

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Set connection policy to Auto + SmartConfig (Device's default connection policy) \*/

// retVal = sl\_WlanPolicySet(SL\_POLICY\_CONNECTION, SL\_CONNECTION\_POLICY(1, 0, 0, 0, 1), NULL, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Remove all profiles \*/

// retVal = sl\_WlanProfileDel(0xFF);

// ASSERT\_ON\_ERROR(retVal);

//

// /\*

// \* Device in station-mode. Disconnect previous connection if any

// \* The function returns 0 if 'Disconnected done', negative number if already disconnected

// \* Wait for 'disconnection' event if 0 is returned, Ignore other return-codes

// \*/

// retVal = sl\_WlanDisconnect();

// if(0 == retVal)

// {

// /\* Wait \*/

// while(IS\_CONNECTED(g\_Status)) { \_SlNonOsMainLoopTask(); }

// }

//

// /\* Enable DHCP client\*/

// retVal = sl\_NetCfgSet(SL\_IPV4\_STA\_P2P\_CL\_DHCP\_ENABLE,1,1,&val);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Disable scan \*/

// configOpt = SL\_SCAN\_POLICY(0);

// retVal = sl\_WlanPolicySet(SL\_POLICY\_SCAN , configOpt, NULL, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Set Tx power level for station mode

// Number between 0-15, as dB offset from max power - 0 will set maximum power \*/

// power = 0;

// retVal = sl\_WlanSet(SL\_WLAN\_CFG\_GENERAL\_PARAM\_ID, WLAN\_GENERAL\_PARAM\_OPT\_STA\_TX\_POWER, 1, (\_u8 \*)&power);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Set PM policy to normal \*/

// retVal = sl\_WlanPolicySet(SL\_POLICY\_PM , SL\_NORMAL\_POLICY, NULL, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Unregister mDNS services \*/

// retVal = sl\_NetAppMDNSUnRegisterService(0, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Remove all 64 filters (8\*8) \*/

// pal\_Memset(RxFilterIdMask.FilterIdMask, 0xFF, 8);

// retVal = sl\_WlanRxFilterSet(SL\_REMOVE\_RX\_FILTER, (\_u8 \*)&RxFilterIdMask,

// sizeof(\_WlanRxFilterOperationCommandBuff\_t));

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = initializeAppVariables();

// ASSERT\_ON\_ERROR(retVal);

//

// return retVal; /\* Success \*/

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief Connecting to a WLAN Access point

//

// This function connects to the required AP (SSID\_NAME).

// The function will return once we are connected and have acquired IP address

//

// \param[in] None

//

// \return 0 on success, negative error-code on error

//

// \note

//

// \warning If the WLAN connection fails or we don't acquire an IP address,

// We will be stuck in this function forever.

//\*/

//static \_i32 establishConnectionWithAP()

//{

// SlSecParams\_t secParams = {0};

// \_i32 retVal = 0;

//

// secParams.Key = PASSKEY;

// secParams.KeyLen = pal\_Strlen(secParams.Key);

// secParams.Type = SEC\_TYPE;

//

// retVal = sl\_WlanConnect(SSID\_NAME, pal\_Strlen(SSID\_NAME), 0, &secParams, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Wait \*/

// while((!IS\_CONNECTED(g\_Status)) || (!IS\_IP\_ACQUIRED(g\_Status))) { \_SlNonOsMainLoopTask(); }

//

// return SUCCESS;

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief Disconnecting from a WLAN Access point

//

// This function disconnects from the connected AP

//

// \param[in] None

//

// \return none

//

// \note

//

// \warning If the WLAN disconnection fails, we will be stuck in this function forever.

//\*/

//static \_i32 disconnectFromAP()

//{

// \_i32 retVal = -1;

//

// /\*

// \* The function returns 0 if 'Disconnected done', negative number if already disconnected

// \* Wait for 'disconnection' event if 0 is returned, Ignore other return-codes

// \*/

// retVal = sl\_WlanDisconnect();

// if(0 == retVal)

// {

// /\* Wait \*/

// while(IS\_CONNECTED(g\_Status)) { \_SlNonOsMainLoopTask(); }

// }

//

// return SUCCESS;

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function initializes the application variables

//

// \param[in] None

//

// \return 0 on success, negative error-code on error

//\*/

//unsigned short usiEmail1OrTime2 = 0;

//static \_i32 initializeAppVariables()

//{

// g\_Status = 0;

// pal\_Memset(&appData, 0, sizeof(appData)); //This line not in email wifi call

//

// return SUCCESS;

//}

//LH-8-1-2020 copied from get-time-LH, duplicate in email.c above

///\*!

// \brief This function displays the application's banner

//

// \param None

//

// \return None

//\*/

//static void displayBanner()

//{

// CLI\_Write((\_u8 \*)"\n\r\n\r"); //LH Line 0 = blank

// CLI\_Write((\_u8 \*)" Get time application - Version "); //LH line 1

// CLI\_Write(APPLICATION\_VERSION); //LH last half of line 1

// CLI\_Write((\_u8 \*)"\n\r\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r"); //LH Line 2 \*\*\* under top line

//}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*Functions from TI http\_client example

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//LH 4-17-2023 100% Identical to existing Get Time and Email function, so commented out

///\*

// \* ASYNCHRONOUS EVENT HANDLERS -- Start

// \*/

//

///\*!

// \brief This function handles WLAN events

//

// \param[in] pWlanEvent is the event passed to the handler

//

// \return None

//

// \note

//

// \warning

//\*/

//void SimpleLinkWlanEventHandler(SlWlanEvent\_t \*pWlanEvent)

//{

// if(pWlanEvent == NULL)

// {

// CLI\_Write(" [WLAN EVENT] NULL Pointer Error \n\r");

// return;

// }

//

// switch(pWlanEvent->Event)

// {

// case SL\_WLAN\_CONNECT\_EVENT:

// {

// SET\_STATUS\_BIT(g\_Status, STATUS\_BIT\_CONNECTION);

//

// /\*

// \* Information about the connected AP (like name, MAC etc) will be

// \* available in 'slWlanConnectAsyncResponse\_t' - Applications

// \* can use it if required

// \*

// \* slWlanConnectAsyncResponse\_t \*pEventData = NULL;

// \* pEventData = &pWlanEvent->EventData.STAandP2PModeWlanConnected;

// \*

// \*/

// }

// break;

//

// case SL\_WLAN\_DISCONNECT\_EVENT:

// {

// slWlanConnectAsyncResponse\_t\* pEventData = NULL;

//

// CLR\_STATUS\_BIT(g\_Status, STATUS\_BIT\_CONNECTION);

// CLR\_STATUS\_BIT(g\_Status, STATUS\_BIT\_IP\_ACQUIRED);

//

// pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;

//

// /\* If the user has initiated 'Disconnect' request, 'reason\_code' is SL\_USER\_INITIATED\_DISCONNECTION \*/

// if(SL\_WLAN\_DISCONNECT\_USER\_INITIATED\_DISCONNECTION == pEventData->reason\_code)

// {

// CLI\_Write(" Device disconnected from the AP on application's request \n\r");

// }

// else

// {

// CLI\_Write(" Device disconnected from the AP on an ERROR..!! \n\r");

// }

// }

// break;

//

// default:

// {

// CLI\_Write(" [WLAN EVENT] Unexpected event \n\r");

// }

// break;

// }

//}

//LH 4-17-2023 100% Identical to existing Get Time and Email function, so commented out

///\*!

// \brief This function handles events for IP address acquisition via DHCP

// indication

//

// \param[in] pNetAppEvent is the event passed to the handler

//

// \return None

//

// \note

//

// \warning

//\*/

//void SimpleLinkNetAppEventHandler(SlNetAppEvent\_t \*pNetAppEvent)

//{

// if(pNetAppEvent == NULL)

// {

// CLI\_Write(" [NETAPP EVENT] NULL Pointer Error \n\r");

// return;

// }

//

// switch(pNetAppEvent->Event)

// {

// case SL\_NETAPP\_IPV4\_IPACQUIRED\_EVENT:

// {

// SET\_STATUS\_BIT(g\_Status, STATUS\_BIT\_IP\_ACQUIRED);

//

// /\*

// \* Information about the connected AP's IP, gateway, DNS etc

// \* will be available in 'SlIpV4AcquiredAsync\_t' - Applications

// \* can use it if required

// \*

// \* SlIpV4AcquiredAsync\_t \*pEventData = NULL;

// \* pEventData = &pNetAppEvent->EventData.ipAcquiredV4;

// \* <gateway\_ip> = pEventData->gateway;

// \*

// \*/

// }

// break;

//

// default:

// {

// CLI\_Write(" [NETAPP EVENT] Unexpected event \n\r");

// }

// break;

// }

//}

//LH 4-17-2023 100% Identical to existing Get Time and Email function, so commented out

///\*!

// \brief This function handles callback for the HTTP server events

//

// \param[in] pHttpEvent - Contains the relevant event information

// \param[in] pHttpResponse - Should be filled by the user with the

// relevant response information

//

// \return None

//

// \note

//

// \warning

//\*/

//void SimpleLinkHttpServerCallback(SlHttpServerEvent\_t \*pHttpEvent,

// SlHttpServerResponse\_t \*pHttpResponse)

//{

// /\*

// \* This application doesn't work with HTTP server - Hence these

// \* events are not handled here

// \*/

// CLI\_Write(" [HTTP EVENT] Unexpected event \n\r");

//}

//LH 4-17-2023 100% Identical to existing Get Time and Email function, so commented out

///\*!

// \brief This function handles general error events indication

//

// \param[in] pDevEvent is the event passed to the handler

//

// \return None

//\*/

//void SimpleLinkGeneralEventHandler(SlDeviceEvent\_t \*pDevEvent)

//{

// /\*

// \* Most of the general errors are not FATAL are are to be handled

// \* appropriately by the application

// \*/

// CLI\_Write(" [GENERAL EVENT] \n\r");

//}

//LH 4-17-2023 100% Identical to existing Get Time and Email function, so commented out

///\*!

// \brief This function handles socket events indication

//

// \param[in] pSock is the event passed to the handler

//

// \return None

//\*/

//void SimpleLinkSockEventHandler(SlSockEvent\_t \*pSock)

//{

// if(pSock == NULL)

// {

// CLI\_Write(" [SOCK EVENT] NULL Pointer Error \n\r");

// return;

// }

//

// switch( pSock->Event )

// {

// case SL\_SOCKET\_TX\_FAILED\_EVENT:

// {

// /\*

// \* TX Failed

// \*

// \* Information about the socket descriptor and status will be

// \* available in 'SlSockEventData\_t' - Applications can use it if

// \* required

// \*

// \* SlSockEventData\_u \*pEventData = NULL;

// \* pEventData = & pSock->socketAsyncEvent;

// \*/

// switch( pSock->socketAsyncEvent.SockTxFailData.status )

// {

// case SL\_ECLOSE:

// CLI\_Write(" [SOCK EVENT] Close socket operation failed to transmit all queued packets\n\r");

// break;

//

//

// default:

// CLI\_Write(" [SOCK EVENT] Unexpected event \n\r");

// break;

// }

// }

// break;

//

// default:

// CLI\_Write(" [SOCK EVENT] Unexpected event \n\r");

// break;

// }

//}

/\*

\* ASYNCHRONOUS EVENT HANDLERS -- End

\*/

//4-18-2023, not used for IFTTT http-put to send email

///\*!

// \brief This function demonstrate the HTTP POST method

//

// \param[in] httpClient - HTTP Client object

//

// \return 0 on success else -ve

//

// \note

//

// \warning

//\*/

//static \_i32 HTTPPostMethod(HTTPCli\_Handle httpClient)

//{

// \_i32 retVal = 0;

// \_i8 tmpBuf[4];

// bool moreFlags = 1;

// bool lastFlag = 1;

// const HTTPCli\_Field fields[4] = {

// {HTTPCli\_FIELD\_NAME\_HOST, HOST\_NAME},

// {HTTPCli\_FIELD\_NAME\_ACCEPT, "\*/\*"},

// {HTTPCli\_FIELD\_NAME\_CONTENT\_TYPE, "application/json"},

// {NULL, NULL}

// };

//

// /\* Set request header fields to be send for HTTP request. \*/

// HTTPCli\_setRequestFields(httpClient, fields);

//

// /\* Send POST method request. \*/

// /\* Here we are setting moreFlags = 1 as there are some more header fields need to send

// other than setted in previous call HTTPCli\_setRequestFields() at later stage.

// Please refer HTTP Library API documentaion @ref HTTPCli\_sendRequest for more information.

// \*/

// moreFlags = 1;

// retVal = HTTPCli\_sendRequest(httpClient, HTTPCli\_METHOD\_POST, POST\_REQUEST\_URI, moreFlags);

// if(retVal < 0)

// {

// CLI\_Write(" Failed to send HTTP POST request header.\n\r");

// return retVal;

// }

//

// sprintf((char \*)tmpBuf, "%d", (sizeof(POST\_DATA)-1));

//

// /\* Here we are setting lastFlag = 1 as it is last header field.

// Please refer HTTP Library API documentaion @ref HTTPCli\_sendField for more information.

// \*/

// lastFlag = 1;

// retVal = HTTPCli\_sendField(httpClient, HTTPCli\_FIELD\_NAME\_CONTENT\_LENGTH, (const char \*)tmpBuf, lastFlag);

// if(retVal < 0)

// {

// CLI\_Write(" Failed to send HTTP POST request header.\n\r");

// return retVal;

// }

//

//

// /\* Send POST data/body \*/

// retVal = HTTPCli\_sendRequestBody(httpClient, POST\_DATA, (sizeof(POST\_DATA)-1));

// if(retVal < 0)

// {

// CLI\_Write(" Failed to send HTTP POST request body.\n\r");

// return retVal;

// }

//

//

// retVal = readResponse(httpClient);

//

// return retVal;

//}

//4-18-2023, not used for IFTTT http-put to send email

///\*!

// \brief This function demonstrate the HTTP DLETE method

//

// \param[in] httpClient - HTTP Client object

//

// \return 0 on success else -ve

//

// \note

//

// \warning

//\*/

//static \_i32 HTTPDeleteMethod(HTTPCli\_Handle httpClient)

//{

// \_i32 retVal = 0;

// bool moreFlags;

// const HTTPCli\_Field fields[3] = {

// {HTTPCli\_FIELD\_NAME\_HOST, HOST\_NAME},

// {HTTPCli\_FIELD\_NAME\_ACCEPT, "\*/\*"},

// {NULL, NULL}

// };

//

//

// /\* Set request header fields to be send for HTTP request. \*/

// HTTPCli\_setRequestFields(httpClient, fields);

//

// /\* Send DELETE method request. \*/

// /\* Here we are setting moreFlags = 0 as there are no more header fields needs to be send

// at later stage. Please refer HTTP Library API documentaion @ref HTTPCli\_sendRequest

// for more information.

// \*/

// moreFlags = 0;

// retVal = HTTPCli\_sendRequest(httpClient, HTTPCli\_METHOD\_DELETE, DELETE\_REQUEST\_URI, moreFlags);

// if(retVal < 0)

// {

// CLI\_Write(" Failed to send HTTP DELETE request header.\n\r");

// return retVal;

// }

//

// retVal = readResponse(httpClient);

//

// return retVal;

//}

/\*!

\brief This function demonstrate the HTTP PUT method

\param[in] httpClient - HTTP Client object

\return 0 on success else -ve

\note

\warning

\*/

**static** \_i32 **HTTPPutMethod**(HTTPCli\_Handle httpClient)

{

\_i32 retVal = 0;

\_i8 tmpBuf[4];

bool moreFlags = 1;

bool lastFlag = 1;

**const** HTTPCli\_Field fields[4] = {

{HTTPCli\_FIELD\_NAME\_HOST, HOST\_NAME},

{HTTPCli\_FIELD\_NAME\_ACCEPT, "\*/\*"},

{NULL, NULL}

};

/\* Set request header fields to be send for HTTP request. \*/

**HTTPCli\_setRequestFields**(httpClient, fields);

/\* Send PUT method request. \*/

/\* Here we are setting moreFlags = 1 as there are some more header fields needs to be send

other than setted in previous call HTTPCli\_setRequestFields() at later stage.

Please refer HTTP Library API documentation @ref HTTPCli\_sendRequest for more information.

\*/

moreFlags = 1;

retVal = **HTTPCli\_sendRequest**(httpClient, HTTPCli\_METHOD\_PUT, PUT\_REQUEST\_URI, moreFlags);

**if**(retVal < 0)

{

CLI\_Write(" Failed to send HTTP PUT request header.\n\r");

**return** retVal;

}

//4-20-2023 getting compile error and seems to be regarding sprintf, so comment out for now

**sprintf**((**char** \*)tmpBuf, "%d", (**sizeof**(PUT\_DATA)-1));

/\* Here we are setting lastFlag = 1 as it is last header field.

Please refer HTTP Library API documentation @ref HTTPCli\_sendField for more information.

\*/

lastFlag = 1;

retVal = **HTTPCli\_sendField**(httpClient, HTTPCli\_FIELD\_NAME\_CONTENT\_LENGTH, (**char** \*)tmpBuf, lastFlag);

**if**(retVal < 0)

{

CLI\_Write(" Failed to send HTTP PUT request header.\n\r");

**return** retVal;

}

/\* Send PUT data/body \*/

retVal = **HTTPCli\_sendRequestBody**(httpClient, PUT\_DATA, (**sizeof**(PUT\_DATA)-1));

**if**(retVal < 0)

{

CLI\_Write(" Failed to send HTTP PUT request body.\n\r");

**return** retVal;

}

retVal = readResponse(httpClient);

**return** retVal;

}

//4-18-2023, not used for IFTTT http-put to send email

///\*!

// \brief This function demonstarte the HTTP GET method

//

// \param[in] httpClient - HTTP Client object

//

// \return 0 on success else -ve

//

// \note

//

// \warning

//\*/

//static \_i32 HTTPGetMethod(HTTPCli\_Handle httpClient)

//{

// \_i32 retVal = 0;

// bool moreFlags;

// const HTTPCli\_Field fields[4] = {

// {HTTPCli\_FIELD\_NAME\_HOST, HOST\_NAME},

// {HTTPCli\_FIELD\_NAME\_ACCEPT, "\*/\*"},

// {HTTPCli\_FIELD\_NAME\_CONTENT\_LENGTH, "0"},

// {NULL, NULL}

// };

//

//

// /\* Set request header fields to be send for HTTP request. \*/

// HTTPCli\_setRequestFields(httpClient, fields);

//

// /\* Send GET method request. \*/

// /\* Here we are setting moreFlags = 0 as there are no more header fields need to send

// at later stage. Please refer HTTP Library API documentaion @ HTTPCli\_sendRequest

// for more information.

// \*/

// moreFlags = 0;

// retVal = HTTPCli\_sendRequest(httpClient, HTTPCli\_METHOD\_GET, GET\_REQUEST\_URI, moreFlags);

// if(retVal < 0)

// {

// CLI\_Write(" Failed to send HTTP GET request.\n\r");

// return retVal;

// }

//

//

// retVal = readResponse(httpClient);

//

// return retVal;

//}

/\*!

\brief This function read repose from server and dump on console

\param[in] httpClient - HTTP Client object

\return 0 on success else -ve

\note

\warning

\*/

**static** \_i32 **readResponse**(HTTPCli\_Handle httpClient)

{

\_i32 retVal = 0;

\_i32 bytesRead = 0;

\_i32 id = 0;

\_u32 len = 0;

\_i32 json = 0;

\_i8 \*dataBuffer=NULL;

bool moreFlags = 1;

**const** \_i8 \*ids[4] = {

HTTPCli\_FIELD\_NAME\_CONTENT\_LENGTH,

HTTPCli\_FIELD\_NAME\_CONNECTION,

HTTPCli\_FIELD\_NAME\_CONTENT\_TYPE,

NULL

};

/\* Read HTTP POST request status code \*/

retVal = **HTTPCli\_getResponseStatus**(httpClient);

**if**(retVal > 0)

{

**switch**(retVal)

{

**case** 200:

{

CLI\_Write(" HTTP Status 200\n\r"); //4-24-2023 this is printed

/\*

Set response header fields to filter response headers. All

other than set by this call we be skipped by library.

\*/

**HTTPCli\_setResponseFields**(httpClient, (**const** **char** \*\*)ids);

/\* Read filter response header and take appropriate action. \*/

/\* Note:

1. id will be same as index of fileds in filter array setted

in previous HTTPCli\_setResponseFields() call.

2. moreFlags will be set to 1 by HTTPCli\_getResponseField(), if field

value could not be completely read. A subsequent call to

HTTPCli\_getResponseField() will read remaining field value and will

return HTTPCli\_FIELD\_ID\_DUMMY. Please refer HTTP Client Libary API

documenation @ref HTTPCli\_getResponseField for more information.

\*/

**while**((id = **HTTPCli\_getResponseField**(httpClient, (**char** \*)g\_buff, **sizeof**(g\_buff), &moreFlags))

!= HTTPCli\_FIELD\_ID\_END)

{

**switch**(id)

{

**case** 0: /\* HTTPCli\_FIELD\_NAME\_CONTENT\_LENGTH \*/

{

len = **strtoul**((**char** \*)g\_buff, NULL, 0);

}

**break**;

**case** 1: /\* HTTPCli\_FIELD\_NAME\_CONNECTION \*/

{

}

**break**;

**case** 2: /\* HTTPCli\_FIELD\_NAME\_CONTENT\_TYPE \*/

{

**if**(!**strncmp**((**const** **char** \*)g\_buff, "application/json",

**sizeof**("application/json")))

{

json = 1;

}

**else**

{

/\* Note:

Developers are advised to use appropriate

content handler. In this example all content

type other than json are treated as plain text.

\*/

json = 0;

}

CLI\_Write(" ");

CLI\_Write(HTTPCli\_FIELD\_NAME\_CONTENT\_TYPE);

CLI\_Write(" : ");

CLI\_Write("application/json\n\r"); // 4-24-2023 this prints

}

**break**;

**default**:

{

CLI\_Write(" Wrong filter id\n\r");

retVal = -1;

**goto** end;

}

}

}

bytesRead = 0;

**if**(len > **sizeof**(g\_buff))

{

dataBuffer = (\_i8 \*) **malloc**(len);

**if**(dataBuffer)

{

CLI\_Write(" Failed to allocate memory\n\r");

retVal = -1;

**goto** end;

}

}

**else**

{

dataBuffer = (\_i8 \*)g\_buff;

}

/\* Read response data/body \*/

/\* Note:

moreFlag will be set to 1 by HTTPCli\_readResponseBody() call, if more

data is available Or in other words content length > length of buffer.

The remaining data will be read in subsequent call to HTTPCli\_readResponseBody().

Please refer HTTP Client Libary API documenation @ref HTTPCli\_readResponseBody

for more information

\*/

bytesRead = **HTTPCli\_readResponseBody**(httpClient, (**char** \*)dataBuffer, len, &moreFlags);

**if**(bytesRead < 0)

{

CLI\_Write(" Failed to received response body\n\r");

retVal = bytesRead;

**goto** end;

}

**else** **if**( bytesRead < len || moreFlags)

{

CLI\_Write(" Mismatch in content length and received data length\n\r");

**goto** end;

}

dataBuffer[bytesRead] = '\0';

**if**(json)

{

/\* Parse JSON data \*/

retVal = ParseJSONData(dataBuffer);

**if**(retVal < 0)

{

**goto** end;

}

}

**else**

{

/\* treating data as a plain text \*/

}

}

**break**;

**case** 404:

CLI\_Write(" File not found. \r\n");

/\* Handle response body as per requirement.

Note:

Developers are advised to take appopriate action for HTTP

return status code else flush the response body.

In this example we are flushing response body in default

case for all other than 200 HTTP Status code.

\*/

**default**:

/\* Note:

Need to flush received buffer explicitly as library will not do

for next request.Apllication is responsible for reading all the

data.

\*/

FlushHTTPResponse(httpClient);

**break**;

}

}

**else**

{

CLI\_Write(" Failed to receive data from server.\r\n");

**goto** end;

}

retVal = 0;

end:

**if**(len > **sizeof**(g\_buff) && (dataBuffer != NULL))

{

**free**(dataBuffer);

}

**return** retVal;

}

/\*!

\brief This function establish a HTTP connection

\param[in] httpClient - HTTP Client object

\return 0 on success else -ve

\note

\warning

\*/

**static** \_i32 **ConnectToHTTPServer**(HTTPCli\_Handle httpClient)

{

\_i32 retVal = -1;

**struct** sockaddr\_in addr;

**#ifdef** USE\_PROXY

**struct** sockaddr\_in paddr;

paddr.sin\_family = AF\_INET;

paddr.sin\_port = htons(PROXY\_PORT);

paddr.sin\_addr.s\_addr = sl\_Htonl(PROXY\_IP);

HTTPCli\_setProxy((**struct** sockaddr \*)&paddr);

**#endif**

/\* Resolve HOST NAME/IP \*/

retVal = sl\_NetAppDnsGetHostByName(HOST\_NAME, pal\_Strlen(HOST\_NAME),

&g\_DestinationIP, SL\_AF\_INET);

**if**(retVal < 0)

{

CLI\_Write(" Device couldn't get the IP for the host-name\r\n");

ASSERT\_ON\_ERROR(retVal);

}

/\* Set up the input parameters for HTTP Connection \*/

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(HOST\_PORT);

addr.sin\_addr.s\_addr = sl\_Htonl(g\_DestinationIP);

/\* HTTPCli open call: handle, address params only \*/

**HTTPCli\_construct**(httpClient);

retVal = **HTTPCli\_connect**(httpClient, (**struct** sockaddr \*)&addr, 0, NULL);

**if** (retVal < 0)

{

CLI\_Write("Connection to server failed\n\r");

ASSERT\_ON\_ERROR(retVal);

}

CLI\_Write(" Successfully connected to the server \r\n");

**return** SUCCESS;

}

/\*!

\brief This function flush HTTP response

\param[in] httpClient - HTTP Client object

\return None

\note

\warning

\*/

**static** **void** **FlushHTTPResponse**(HTTPCli\_Handle httpClient)

{

**const** \_i8 \*ids[2] = {

HTTPCli\_FIELD\_NAME\_CONNECTION,

NULL

};

\_i8 buf[128];

\_i32 id;

\_i32 len = 1;

bool moreFlag = 0;

\_i8 \*\*prevRespFilelds = NULL;

/\* Store previosly store array if any \*/

prevRespFilelds = (\_i8 \*\*)**HTTPCli\_setResponseFields**(httpClient, (**const** **char** \*\*)ids);

/\* Read response headers \*/

**while** ((id = **HTTPCli\_getResponseField**(httpClient, (**char** \*)buf, **sizeof**(buf), &moreFlag))

!= HTTPCli\_FIELD\_ID\_END)

{

**if**(id == 0)

{

**if**(!**strncmp**((**const** **char** \*)buf, "close", **sizeof**("close")))

{

CLI\_Write(" Connection terminated by server\n\r");

}

}

}

/\* Restore previously store array if any \*/

**HTTPCli\_setResponseFields**(httpClient, (**const** **char** \*\*)prevRespFilelds);

**while**(1)

{

/\* Read response data/body \*/

/\* Note:

moreFlag will be set to 1 by HTTPCli\_readResponseBody() call, if more

data is available Or in other words content length > length of buffer.

The remaining data will be read in subsequent call to HTTPCli\_readResponseBody().

Please refer HTTP Client Libary API documenation @ref HTTPCli\_readResponseBody

for more information.

\*/

**HTTPCli\_readResponseBody**(httpClient, (**char** \*)buf, **sizeof**(buf) - 1, &moreFlag);

CLI\_Write((\_u8 \*)buf);

CLI\_Write("\r\n");

**if** ((len - 2) >= 0 && buf[len - 2] == '\r' && buf [len - 1] == '\n')

{

}

**if**(!moreFlag)

{

/\* There no more data. break the loop. \*/

**break**;

}

}

}

/\*!

\brief This function parse JSON data

\param[in] ptr - Pointer to JSON data

\return 0 on success else -ve

\note

\warning

\*/

**static** \_i32 **ParseJSONData**(\_i8 \*ptr)

{

\_i32 retVal = 0;

\_i32 noOfToken;

jsmn\_parser parser;

jsmntok\_t \*tokenList;

\_i8 printBuffer[4];

/\* Initialize JSON PArser \*/

**jsmn\_init**(&parser);

/\* Get number of JSON token in stream as we we don't know how many tokens need to pass \*/

noOfToken = **jsmn\_parse**(&parser, (**const** **char** \*)ptr, **strlen**((**const** **char** \*)ptr), NULL, 10);

**if**(noOfToken <= 0)

{

CLI\_Write(" Failed to initialize JSON parser\n\r");

**return** -1;

}

/\* Allocate memory to store token \*/

tokenList = (jsmntok\_t \*) **malloc**(noOfToken\***sizeof**(jsmntok\_t));

**if**(tokenList == NULL)

{

CLI\_Write(" Failed to allocate memory\n\r");

**return** -1;

}

/\* Initialize JSON Parser again \*/

**jsmn\_init**(&parser);

noOfToken = **jsmn\_parse**(&parser, (**const** **char** \*)ptr, **strlen**((**const** **char** \*)ptr), tokenList, noOfToken);

**if**(noOfToken < 0)

{

CLI\_Write(" Failed to parse JSON tokens\n\r");

retVal = noOfToken;

}

**else**

{

CLI\_Write(" Successfully parsed ");

//4-20-2023 getting compile error and seems to be regarding sprintf, so comment out for now

//sprintf stands for “String print”. Instead of printing on console, it store output on char buffer which are specified in sprintf.

// sprintf((char \*)printBuffer, "%ld", noOfToken);

CLI\_Write((\_u8 \*)printBuffer);

CLI\_Write(" JSON tokens\n\r");

}

**free**(tokenList);

**return** retVal;

}

// 4-17-2023 LH 100% identical to function above for static \_i32 configureSimpleLinkToDefaultState()

// So Commented out to remove duplication

///\*!

// \brief Obtain the file from the server

//

// This function requests the file from the server and save it on serial flash.

// To request a different file for different user needs to modify the

// PREFIX\_BUFFER and POST\_BUFFER macros.

//

// \param[in] None

//

// \return 0 for success and negative for error

//

//\*/

///\*!

// \brief This function configure the SimpleLink device in its default state. It:

// - Sets the mode to STATION

// - Configures connection policy to Auto and AutoSmartConfig

// - Deletes all the stored profiles

// - Enables DHCP

// - Disables Scan policy

// - Sets Tx power to maximum

// - Sets power policy to normal

// - Unregisters mDNS services

// - Remove all filters

//

// \param[in] none

//

// \return On success, zero is returned. On error, negative is returned

//\*/

//static \_i32 configureSimpleLinkToDefaultState()

//{

// SlVersionFull ver = {0};

// \_WlanRxFilterOperationCommandBuff\_t RxFilterIdMask = {0};

//

// \_u8 val = 1;

// \_u8 configOpt = 0;

// \_u8 configLen = 0;

// \_u8 power = 0;

//

// \_i32 retVal = -1;

// \_i32 mode = -1;

//

// mode = sl\_Start(0, 0, 0);

// ASSERT\_ON\_ERROR(mode);

//

// /\* If the device is not in station-mode, try configuring it in station-mode \*/

// if (ROLE\_STA != mode)

// {

// if (ROLE\_AP == mode)

// {

// /\* If the device is in AP mode, we need to wait for this event before doing anything \*/

// while(!IS\_IP\_ACQUIRED(g\_Status)) { \_SlNonOsMainLoopTask(); }

// }

//

// /\* Switch to STA role and restart \*/

// retVal = sl\_WlanSetMode(ROLE\_STA);

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = sl\_Start(0, 0, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Check if the device is in station again \*/

// if (ROLE\_STA != retVal)

// {

// /\* We don't want to proceed if the device is not coming up in station-mode \*/

// ASSERT\_ON\_ERROR(DEVICE\_NOT\_IN\_STATION\_MODE);

// }

// }

//

// /\* Get the device's version-information \*/

// configOpt = SL\_DEVICE\_GENERAL\_VERSION;

// configLen = sizeof(ver);

// retVal = sl\_DevGet(SL\_DEVICE\_GENERAL\_CONFIGURATION, &configOpt, &configLen, (\_u8 \*)(&ver));

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Set connection policy to Auto + SmartConfig (Device's default connection policy) \*/

// retVal = sl\_WlanPolicySet(SL\_POLICY\_CONNECTION, SL\_CONNECTION\_POLICY(1, 0, 0, 0, 1), NULL, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Remove all profiles \*/

// retVal = sl\_WlanProfileDel(0xFF);

// ASSERT\_ON\_ERROR(retVal);

//

// /\*

// \* Device in station-mode. Disconnect previous connection if any

// \* The function returns 0 if 'Disconnected done', negative number if already disconnected

// \* Wait for 'disconnection' event if 0 is returned, Ignore other return-codes

// \*/

// retVal = sl\_WlanDisconnect();

// if(0 == retVal)

// {

// /\* Wait \*/

// while(IS\_CONNECTED(g\_Status)) { \_SlNonOsMainLoopTask(); }

// }

//

// /\* Enable DHCP client\*/

// retVal = sl\_NetCfgSet(SL\_IPV4\_STA\_P2P\_CL\_DHCP\_ENABLE,1,1,&val);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Disable scan \*/

// configOpt = SL\_SCAN\_POLICY(0);

// retVal = sl\_WlanPolicySet(SL\_POLICY\_SCAN , configOpt, NULL, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Set Tx power level for station mode

// Number between 0-15, as dB offset from maximum power - 0 will set maximum power \*/

// power = 0;

// retVal = sl\_WlanSet(SL\_WLAN\_CFG\_GENERAL\_PARAM\_ID, WLAN\_GENERAL\_PARAM\_OPT\_STA\_TX\_POWER, 1, (\_u8 \*)&power);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Set PM policy to normal \*/

// retVal = sl\_WlanPolicySet(SL\_POLICY\_PM , SL\_NORMAL\_POLICY, NULL, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Unregister mDNS services \*/

// retVal = sl\_NetAppMDNSUnRegisterService(0, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Remove all 64 filters (8\*8) \*/

// pal\_Memset(RxFilterIdMask.FilterIdMask, 0xFF, 8);

// retVal = sl\_WlanRxFilterSet(SL\_REMOVE\_RX\_FILTER, (\_u8 \*)&RxFilterIdMask,

// sizeof(\_WlanRxFilterOperationCommandBuff\_t));

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = sl\_Stop(SL\_STOP\_TIMEOUT);

// ASSERT\_ON\_ERROR(retVal);

//

// retVal = initializeAppVariables();

// ASSERT\_ON\_ERROR(retVal);

//

// return retVal; /\* Success \*/

//}

// 4-17-2023 LH Merged into Get Time and Email establishConnectionWithAP

///\*!

// \brief Connecting to a WLAN Access point

//

// This function connects to the required AP (SSID\_NAME).

// The function will return once we are connected and have acquired IP address

//

// \param[in] None

//

// \return 0 on success, negative error-code on error

//

// \note

//

// \warning If the WLAN connection fails or we don't acquire an IP address,

// We will be stuck in this function forever.

//\*/

//static \_i32 establishConnectionWithAP()

//{

// SlSecParams\_t secParams = {0};

// \_i32 retVal = 0;

//

// secParams.Key = PASSKEY;

// secParams.KeyLen = pal\_Strlen(PASSKEY);

// secParams.Type = SEC\_TYPE;

//

// retVal = sl\_WlanConnect(SSID\_NAME, pal\_Strlen(SSID\_NAME), 0, &secParams, 0);

// ASSERT\_ON\_ERROR(retVal);

//

// /\* Wait \*/

// while((!IS\_CONNECTED(g\_Status)) || (!IS\_IP\_ACQUIRED(g\_Status))) { \_SlNonOsMainLoopTask(); }

//

// return SUCCESS;

//}

// 4-17-2023 LH Merged into Get Time and Email intializeAppVariables

///\*!

// \brief This function initializes the application variables

//

// \param[in] None

//

// \return 0 on success, negative error-code on error

//\*/

//static \_i32 initializeAppVariables()

//{

// g\_Status = 0;

// g\_SockID = 0;

// g\_DestinationIP = 0;

// g\_BytesReceived = 0;

// pal\_Memset(g\_buff, 0, sizeof(g\_buff));

//

// return SUCCESS;

//}

//4-17-2023 LH Merged into GetTime and Email Display Banner function call

///\*!

// \brief This function displays the application's banner

//

// \param None

//

// \return None

//\*/

//static void displayBanner()

//{

// CLI\_Write("\n\r\n\r");

// CLI\_Write(" HTTP Client - Version ");

// CLI\_Write(APPLICATION\_VERSION);

// CLI\_Write("\n\r\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");

//}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*End of ......Functions from TI http\_client example

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## emailAndGetTime.h

/\*

\* email.h - email header file

\*

\* Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

\*

\*

\* All rights reserved. Property of Texas Instruments Incorporated.

\* Restricted rights to use, duplicate or disclose this code are

\* granted through contract.

\*

\* The program may not be used without the written permission of

\* Texas Instruments Incorporated or against the terms and conditions

\* stipulated in the agreement under which this program has been supplied,

\* and under no circumstances can it be used with non-TI connectivity device.

\*

\*/

**#include** "simplelink.h"

**#ifndef** \_\_EMAIL\_H\_\_

**#define** \_\_EMAIL\_H\_\_

**#ifdef** \_\_cplusplus

**extern** "C" {

**#endif**

/\* SMTP defines \*/

**#define** SMTP\_BUF\_LEN 100

/\* NetApp Email set/get options \*/

**#define** NETAPP\_ADVANCED\_OPT (1)

**#define** NETAPP\_SOURCE\_EMAIL (2)

**#define** NETAPP\_PASSWORD (3)

**#define** NETAPP\_DEST\_EMAIL (4)

**#define** NETAPP\_SUBJECT (5)

**#define** NETAPP\_MESSAGE (6)

**#define** MAX\_DEST\_EMAIL\_LEN 30

**#define** MAX\_USERNAME\_LEN 30

**#define** MAX\_PASSWORD\_LEN 30

//5-28-2020 LH Change Max subject length to 64

//#define MAX\_SUBJECT\_LEN 30

**#define** MAX\_SUBJECT\_LEN 64

**#define** MAX\_MESSAGE\_LEN 64

**#define** BASEKEY\_LEN 128

**#define** MAX\_EMAIL\_RCF\_LEN (MAX\_DEST\_EMAIL\_LEN + 2)

/\* NetApp Email protocol types \*/

**#define** SL\_NET\_APP\_SMTP\_PROTOCOL (1)

**typedef** **struct**

{

\_u32 ProtocolSubType;

\_u32 Port;

\_u32 Family;

\_u32 SecurityMethod;

\_u32 SecurityCypher;

\_u32 Ip; /\* IPv4 address or IPv6 first 4 bytes \*/

\_u32 Ip1OrPaadding;

\_u32 Ip2OrPaadding;

\_u32 Ip3OrPaadding;

}SlNetAppEmailOpt\_t;

**typedef** **struct**

{

\_u8 Username[MAX\_USERNAME\_LEN];

}SlNetAppSourceEmail\_t;

**typedef** **struct**

{

\_u8 Password[MAX\_PASSWORD\_LEN];

}SlNetAppSourcePassword\_t;

**typedef** **struct**

{

\_u8 Email[MAX\_DEST\_EMAIL\_LEN];

}SlNetAppDestination\_t;

**typedef** **struct**

{

\_u8 Value[MAX\_SUBJECT\_LEN];

}SlNetAppEmailSubject\_t;

//LH 8-2-2020 getting errors that appData is not defined so try pasting it here, I added the word typedef in front to make it the same

//typedef struct{

//Taking the word typedef off gave a clean compile.....why do not know

**struct**{

\_u32 DestinationIP;

\_u32 elapsedSec;

\_u32 uGeneralVar;

\_u32 uGeneralVar1;

\_u16 ccLen;

\_i32 SockID;

\_i32 sGeneralVar;

\_u8 time[30]; //LH time is a structure with 30 elements so it can display a string like this Tue Oct 10 2018 11:26:13

\_u8 \*ccPtr;

}appData;

/\*!

\brief This function handles WLAN events

\param[in] command - Command sent for processing

\param[in] pValueLen - Length of data to be processed

\param[in] pValue - Data to be processed

\return 0 for success, -1 otherwise

\note

\warning

\*/

\_i32 **sl\_NetAppEmailSet**(\_u8 command, \_u8 pValueLen,

\_u8 \*pValue);

/\*!

\brief Create a secure socket and connects to SMTP server

\param[in] none

\return 0 if success and negative in case of error

\note

\warning

\*/

\_i32 **sl\_NetAppEmailConnect**();

/\*!

\brief Checks the connection status and sends the Email

\param[in] none

\return 0 if success otherwise -1

\note

\warning

\*/

\_i32 **sl\_NetAppEmailSend**();

//LH added

//void LHSendEmail (void);

**int** **LHSendEmail** ();

**int** **LHGetTime** (); //8-1-2020

**int** **LHIFTTTSendEamil**(); //4-18-2023

//From myTimers

**extern** **char** LHemailSubject[64];

**extern** **char** LHemailBody[64];

//End of ...//LH added

**#ifdef** \_\_cplusplus

}

**#endif** /\* \_\_cplusplus \*/

**#endif** /\* \_\_NETAPP\_H\_\_ \*/

# The End