IFTTT

# Introduction

This document provides basic information about IFTTT. It discusses using Talaria TWO with IFTTT with the help of an example application. The information includes setting up Talaria TWO to connect to IFTTT, a stepwise procedure to set up IFTTT account and the required infrastructure in IFTTT.

# IFTTT

IFTTT is a web service that provides easiest way to connect thousands of apps and devices, including Google Drive, Amazon Alexa, Fitbit, Twitter, Dropbox, Ring. Using IFTTT, two apps called *services* are connected using an *applet* defined in IFTTT webservice.

## Accessing IFTTT

IFTTT web service can be access using the following link: <https://ifttt.com/>.

## Creating an Account in IFTTT

Having an account in IFTTT web service is a prerequisite to exploring more and getting hands-on with the example application. This is free and simple to create. Open the link in section 3.1 and click on Get Started. Follow the steps as stated on the website.

## IFTTT Concepts

There are four important concepts in IFTTT:

1. Service:

Services are already available/defined interfaces such as Twitter, Facebook, Google Assistant etc.

**Note**: These are IFTTT services defined in the IFTTT platform. They internally connect with actual web services/applications that are available outside IFTTT. Details on creating services is outside the scope of this document.

1. Applet:

Applet is an entity defined in IFTTT that connects two services.

1. Trigger:

Once two IFTTT services are connected using an applet, one of the application acts as the trigger. This IFTTT service that acts as trigger, receives a request from outside and triggers a specific action in the other service connected in the same applet.

1. Action:

When the IFTTT service acting as the trigger triggers the other service in the same applet, an action is taken by the other service.

## Applet

Applet on the IFTTT platform define the following:

1. The services to be connected
2. The service that is going to act as a trigger
3. The service that is going to take action

# Creating an Applet for Talaria TWO

This section discusses steps to create an applet for Talaria TWO to interact with IFTTT. For this, we choose already available IFTTT services like Webhooks and Email. The idea is to send data/request to Webhooks service from Talaria TWO. This should trigger an email being sent to the email address configured in the Email IFTTT service.

**Note**: It is required to have an account before executing the next set of steps.

1. Open maker.ifttt.com and sign in.
2. Click Create in the top right. Select If This.
3. In the search field, enter Webhooks and select that service. Select Receive a web request.
4. In the Event Name field, enter suitable name like <trigger\_from\_t2>. Click the Create trigger button.
5. Now, click on Then That. Select the Email service.
6. Choose the Send me an email action. If this is your first time setting up an email action, click Connect, and proceed with the configuration as instructed on IFTTT.
7. Both the trigger and action have now been configured. Click Continue.
8. Click Finish to create the applet.

With this, an applet is defined.

Now, it is required to get the Webhook endpoint URL. Execute the following steps to acquire the same:

1. Click on your profile picture in the top right, then click My services.
2. Choose webhooks and click on Documentation in the upper right.
3. This gives the endpoint, like the following: <https://maker.ifttt.com/trigger/trigger_from_t2/with/key/c9ebSVVNGeSB1yi6NrVeyl>

# Code Walkthrough

## Application Flow

IFTTT is a web service that provides an easy way to connect thousands of apps and devices, including Google Drive, Amazon Alexa, Fitbit, Twitter, Dropbox, Ring. Using IFTTT, two apps named services are connected using an applet defined in IFTTT webservice.

Following are the steps:

1. Connect to Wi-Fi network.
2. Connect to HTTP server.
3. Get the latest config file from remote server.
4. HTTP get method.
5. HTTP post method.

## Sample Code Walkthrough

A user-defined data struct is created to store the data of IFTTT:

|  |
| --- |
| #define APP\_NAME "IFTTT Demo Application"  #define APP\_VERSION "2.0"  OS\_APPINFO {.stack\_size = 4096};  #define INPUT\_PARAMETER\_URL "url"  #define INPUT\_PARAMETER\_PORT "port"  #define INPUT\_PARAMETER\_SECURED "secured"  #define INPUT\_PARAMETER\_METHOD "method"  #define INPUT\_PARAMETER\_CA "ca\_cert"  #define INPUT\_PARAMETER\_POST\_LEN "post\_len"  #define INPUT\_PARAMETER\_CLIENT\_CERT "client\_cert"  #define INPUT\_PARAMETER\_CLIENT\_KEY "client\_key"  #define INPUT\_PARAMETER\_USE\_CA\_BUNDLE "use\_ca\_bundle"  #define NULL\_STR ""  struct param\_t {  const char \*ssid;  const char \*passphrase;  const char \*url;  const char \*port;  const char \*secured;  const char \*method;  const char \*ca\_cert;  const char \*post\_len;  const char \*client\_cert;  const char \*client\_key;  const char \*use\_ca\_bundle;  };  struct param\_t param;  char default\_port[8];  char default\_secured[8];  char default\_post\_len[8];  struct os\_semaphore app\_wcm\_lock;  int wcm\_connect\_success = 0;  static char host[128];  static char path[128]; |

Following boot arguments are passed:

1. URL, port, secured, ca\_cert, method, post\_len, client\_cert, client\_key value and ca\_bundle.
2. SD service type and proto, and action.

|  |
| --- |
| static int  parse\_boot\_args(void)  {  const char \*np\_path;  int ret = 0;  sprintf(default\_port, "%d", 80);  sprintf(default\_secured, "%d", 0);  sprintf(default\_post\_len, "%d", 32);  param.url = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_URL);  param.port = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_PORT);  param.secured = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_SECURED);  param.ca\_cert = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_CA);  param.method = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_METHOD);  param.post\_len = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_POST\_LEN);  param.client\_cert = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_CLIENT\_CERT);  param.client\_key = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_CLIENT\_KEY);  param.use\_ca\_bundle = os\_get\_boot\_arg\_str(INPUT\_PARAMETER\_USE\_CA\_BUNDLE); |

To connect to a Wi-Fi network, wcm\_create()API from the Wi-Fi Connection Manager is used. Initially, the Wi-Fi network interface is created using wcm\_create().

|  |
| --- |
| wcm\_handle = wcm\_create(NULL); |

wifi\_connect\_to\_network()API, from components library, connects to the Wi-Fi network using the AP credentials provided.

|  |
| --- |
| wifi\_connect\_to\_network(&wcm\_handle, WCM\_CONN\_WAIT\_INFINITE, &wcm\_connected);  if(wcm\_connected != true) {  os\_printf("\n[APP]Error: Failed to connect to WiFi N/w");  goto exit;  } |

Here, the data structure http\_client\_config\_t cfg is used to pass the parameter while opening a HTTP connection with remote server using http\_client\_open such as URL, port, secured, ssl\_cfg, time\_out.

|  |
| --- |
| /\* Connect to HTTP server\*/  http\_client\_config\_t cfg = {0};  http\_client\_handle\_t http\_handle;  memset(&cfg, 0, sizeof(http\_client\_config\_t));  path[0]= '\0';  if(param.url){  os\_printf("\n[APP]URL = %s", param.url);  rval = http\_client\_url\_to\_host(param.url, host, sizeof(host), path,  sizeof(path), &cfg.port);  if(rval < 0){  os\_printf("\n[APP]URL is not proper");  os\_printf("\n\texample URLs:");  os\_printf("\n\t\thttps://maker.ifttt.com/trigger/krg\_door\_open/with/key/c9ebSVVNGeSB1yi6NrVeyl");  }  cfg.hostname = host;  }  if(param.port){/\*If specified explicietly, overide the port specified in URL\*/  cfg.port = atoi(param.port);  }  cfg.secured = atoi(param.secured);  if(cfg.secured) {  if(cfg.secured == 1){  cfg.ssl\_cfg.auth\_mode = SSL\_WRAP\_VERIFY\_NONE;  }else{  cfg.ssl\_cfg.auth\_mode = SSL\_WRAP\_VERIFY\_REQUIRED;  if(!atoi(param.use\_ca\_bundle)){  cfg.ssl\_cfg.ca\_cert.buf = utils\_file\_get(param.ca\_cert,  &cfg.ssl\_cfg.ca\_cert.len);  if(NULL == cfg.ssl\_cfg.ca\_cert.buf){  os\_printf("Error: No CA certificate found. Required");  goto exit;  }  }  }  if(param.client\_cert && strlen(param.client\_cert)){  cfg.ssl\_cfg.client\_cert.buf = utils\_file\_get(param.client\_cert,  &cfg.ssl\_cfg.client\_cert.len);  if(NULL == cfg.ssl\_cfg.client\_cert.buf){  os\_printf("Error: Could not open client certificate\n");  goto exit;  }  }  if(param.client\_key && strlen(param.client\_key)){  cfg.ssl\_cfg.client\_key.buf = utils\_file\_get(param.client\_key,  &cfg.ssl\_cfg.client\_key.len);  if(NULL == cfg.ssl\_cfg.client\_key.buf){  os\_printf("Error: Could not open client key\n");  goto exit;  }  }  cfg.secured = 1;  } |

Data structure http\_client\_resp\_info\_t is used to pass information about the data received from the server when HTTP GET is executed using http\_client\_get API.

|  |
| --- |
| static void  app\_http\_cb(void \* ctx, http\_client\_resp\_info\_t \*resp)  {  static int total\_bytes\_rcvd = 0;  static int hdrs\_printed = 0;  int i;  if(NULL == resp) {  return;  }  if(!hdrs\_printed) {  os\_printf("\n\n[APP]Response:\n%d ----------------------\n", resp->resp\_len);  os\_printf("\n%d", resp->status\_code);  i = 0;  while(resp->resp\_hdrs[i]) {  os\_msleep(10);  os\_printf("\n%s", resp->resp\_hdrs[i]);  i++;  }  os\_printf("\n[APP]Body:\n");  hdrs\_printed = 1;  }  total\_bytes\_rcvd += resp->resp\_len;  for(i = 0; i < resp->resp\_len; i++) {  os\_printf("%c", resp->resp\_body[i]);  }  return;  } |

API http\_client\_open connects to the remote HTTP server. The configuration needed for the connection is passed using http\_client\_config\_t.

|  |
| --- |
| http\_handle = http\_client\_open(&cfg);  if(NULL == http\_handle) {  os\_printf("\n[APP]Error: HTTP connection failed");  goto exit;  }  os\_printf("\n[APP]Succes: HTTP connection done");  /\* Get the latest config file from remote server\*/  http\_client\_set\_req\_hdr(http\_handle, "Host", cfg.hostname); |

This function is used for performing HTTP GET. The HTTP response is provided through the call back. The call back is called multiple times until the whole response is received.

|  |
| --- |
| if(!strcmp(param.method, "get")) {  /\*HTTP get \*/  rval = http\_client\_get(http\_handle, (char \*)path, app\_http\_cb,  NULL, 300);  http\_client\_close(http\_handle);  http\_handle = NULL;  if(rval < 0) {  os\_printf("\n[APP]Failure : http\_client\_get(), rval = %d", rval);  goto exit;  }else{  os\_printf("\n[APP]Success");  }  } |

This function is used to perform HTTP POST. Using this data can be sent to the HTTP server. The response is provided using the call back. Setting content length header is a must using http\_client\_set\_req\_hdr before calling this API.

|  |
| --- |
| if(!strcmp(param.method, "post")) {  /\*HTTP post \*/  char \*post\_data;  int post\_data\_len = 0; /\*atoi(param.post\_len);\*/  int send\_len;  char conetnt\_len\_hdr\_val[16];  post\_data = os\_alloc(1024);  if(NULL == post\_data) {  os\_printf("\n[APP]Error: malloc failre for post\_data");  goto exit;  }  sprintf(conetnt\_len\_hdr\_val, "%d", post\_data\_len);  http\_client\_set\_req\_hdr(http\_handle, "Content-length", conetnt\_len\_hdr\_val);  while(post\_data\_len) {  send\_len = post\_data\_len > 1024 ? 1024 : post\_data\_len;  rval = http\_client\_post(http\_handle, (char \*)path,  post\_data, send\_len,  app\_http\_cb, NULL, 300);  if(rval < 0) {  os\_printf("\n[APP]Failure : http\_client\_post(), rval = %d", rval);  goto exit;  }  post\_data\_len -= send\_len;  }  http\_client\_close(http\_handle);  if(rval >= 0){  os\_printf("\n[APP]Success");  }else  goto exit;  }  os\_printf("\n\n[APP]------ Program Exit-------------\n\n");  return 0;  exit:  os\_printf("\n\n[APP]!!!!!! Error Exit !!!!!!!!!!!!!\n\n");  return 0;  } |

# Running the Application

## Programming Talaria TWO board with ELF

Program ifttt.elf(sdk\_x.y\examples\ifttt\bin) using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down.
   2. ELF Input: Load the ifttt.elf by clicking on Select ELF File.
   3. Boot arguments: Pass the following boot arguments:

|  |
| --- |
| url=https://maker.ifttt.com/trigger/trigger\_from\_t2/with/key/c9ebSVVNGeSB1yi6NrVeyl,port=443,secured=1,method=get |

where,

* + 1. url: Webhooks end point URL
    2. port: 443
    3. secured: 1 - Connecting to server without server verification
    4. method: GET
  1. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y/pc\_tools/Download\_Tool/doc*).

**Note**: x and y refer to the SDK release version. For example: sdk\_2.4/doc.

## Expected Output

When the application is run with a proper endpoint URL, an email is received from IFTTT to the email address provided in the Email service at the time of creating the applet.

|  |
| --- |
| ----------------PROG Flash: Start Time 05 Apr 2023 13:28:13 PM -------------  UART:SNWWWWAE  Build $Id: git-f1a4f00fb $  hio.baudrate=921600  flash: Gordon ready!  Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAEBuild $Id: git-f1a4f00fb $  ssid=tplink\_debug url=https://maker.ifttt.com/trigger/trigger\_from\_t2/with/key/cfYYk8z-DTB0yqc1bykK7ITP3A31mMW7J7QEfy-wbsk port=443 secured=1 method=get passphrase=InnoQA2023$  $App:git-6519235e  SDK Ver: sdk\_2.6.3\_alpha  IFTTT Demo App  Application Information:  ------------------------  Name : IFTTT Demo Application  Version : 2.0  Build Date : Apr 3 2023  Build Time : 20:57:22  Heap Available: 331 KB (339096 Bytes)  [APP]Bootparams :  --------------------  url=https://maker.ifttt.com/trigger/trigger\_from\_t2/with/key/cfYYk8z-DTB0yqc1bykK7ITP3A31mMW7J7QEfy-wbsk  path= 443  secured= 1  method= get  ca\_cert=<null>  post\_len=<null>  use\_ca\_bundle = <null>  [APP]Bootparams end here....  [APP]Bootparams check done....ret = 0  addr e0:69:3a:00:0a:6a  network profile created for ssid: tplink\_debug  Connecting to added network : tplink\_debug  [0.945,385] CONNECT:60:32:b1:3a:83:ee Channel:1 rssi:-51 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [1.217,714] MYIP 192.168.0.155  [1.217,763] IPv6 [fe80::e269:3aff:fe00:a6a]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Connected to added network : tplink\_debug  [APP]Success: Connected to <null> N/w  [APP]URL = https://maker.ifttt.com/trigger/trigger\_from\_t2/with/key/cfYYk8z-DTB0yqc1bykK7ITP3A31mMW7J7QEfy-wbsk  . [SSL\_WRAP]Checking input configurations...  . [SSL\_WRAP]Seeding the random number generator...  . [SSL\_WRAP]Connecting to tcp maker.ifttt.com:443...  . [SSL\_WRAP]Setting up the SSL/TLS structure...  . [SSL\_WRAP]setting configurations..  >auth mode = 0 (0- skip, 1- optional, 2- required  >max fragment len = 0  >Handshake timeout = 30 Sec  . [SSL\_WRAP]Performing the SSL/TLS handshake...  . [SSL\_WRAP] Handshake done. ok  . [SSL\_WRAP]Verifying peer X.509 certificate.  [APP]Succes: HTTP connection done  [APP]Response:  55 ---------------------- 200  Content-Type: text/html; charset=utf-8  Content-Length: 55  Connection: keep-alive  Date: Wed, 05 Apr 2023 07:58:29 GMT  ETag: W/"37-4jb44xEDYSdzHhse3E8QAYlkiaM"  X-Clacks-Overhead: GNU Terry Pratchett  X-Powered-By: Sad Unicorns  X-Robots-Tag: none  X-Top-Secrettt: VG9vIGVhc3k/IElmIHlvdSBjYW4gcmVhZCB0aGlzLCBFbWFpbCB1cyBhdCBqb2JzK3NlY3JldEBpZnR0dC5jb20uIFdlIHdhbnQgTWFrZXJzLg==  X-Cache: Miss from cloudfront  Via: 1.1 936179e33a47597eee0ce6ca36a747ba.cloudfront.net (CloudFront)  X-Amz-Cf-Pop: BLR50-P1  X-Amz-Cf-Id: AnEpXUJbESULCvVfelLrCMWmMFn2AFmHimcNQMFFg\_OTBmbmI4EN1A==  [APP]Body:  Congratulations! You've fired the trigger\_from\_t2 event  [APP]Success  [APP]------ Program Exit------------- |