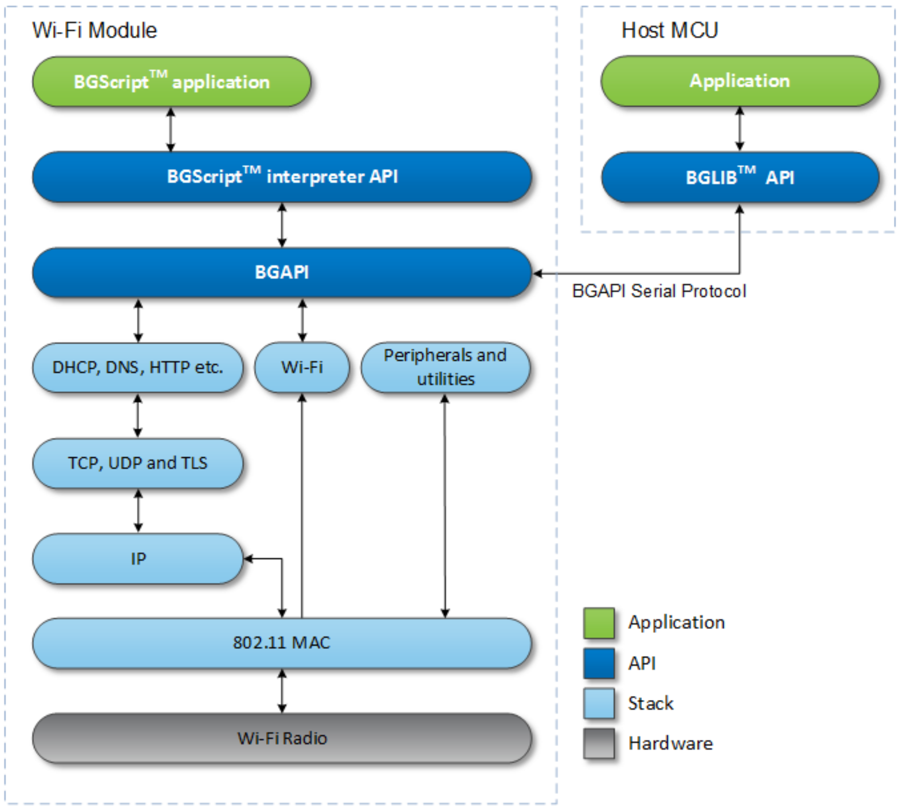
AN974: Wizard Gecko TLS and SMTP Example

This application note introduces the usage of Transport Layer Security TLS/SSL, one of the security features in the WGM110 Wi-Fi® Module. This security protocol makes it possible (for devices embedding a WGM110 Module) to establish a secure network socket to the Internet using security certificates. Wizard Gecko implementation of TLS protocol is based on the X.509 certificates, which need to be properly set and stored to the Module's flash or RAM for the protocol to work.

This application note includes a brief introduction to the WGM110 Wi-Fi stack and a high-level description of the project's functionality and structure, followed by a deeper explanation of each component contained in the project.

KEY POINTS

- A complete application example
- Showcased features
  - Project configuration
  - Hardware configuration
  - X.509 certificates
  - HTTP Server
  - Wi-Fi Station
  - TLS
  - SMTP



1. Introduction

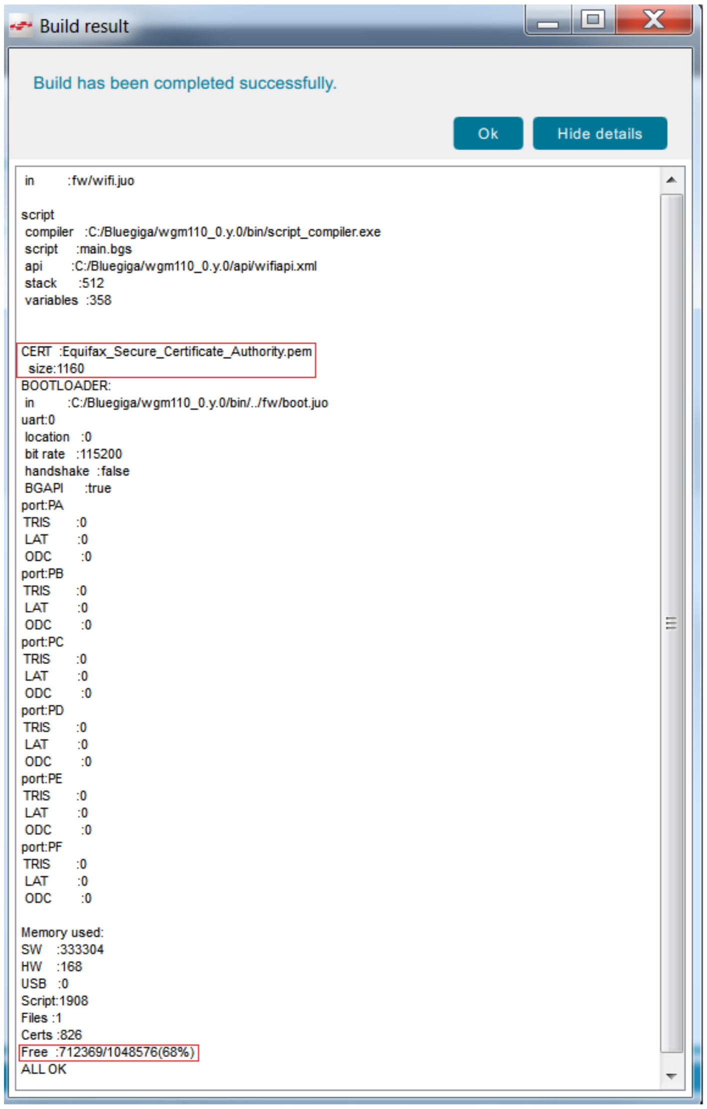
This application note introduces the usage of transport level security TLS/SSL, which is one of the security features in the WGM110. This security protocol makes it possible (for devices embedding a WGM110 Module) to establish a secure network socket to the Internet using security certificates. The TLS protocol uses X.509 certificates which need to be properly set and stored to the Module's flash or RAM for the protocol to work.

The code base for this application note is the *Gmail* example from the Wizard Gecko SDK. In this example, the WGM110 Module establishes a secure connection to *smtp.gmail.com* server and sends a simple **“Hello World”** email when push button **PB0** on the WSTK Main Board is pressed. For the secure connection, a root certificate is required to verify the identity of the server. SMTP protocol is implemented using BGScript, because it is not natively supported in the WGM110 Module.

2. X.509

X.509 is a cryptography standard, managed by International Telecommunications Union Standardization Sector (ITU-T), for public key infrastructure and privilege infrastructure management. The standard specifies formats for public key certificates, certification revocation lists, certification path validation algorithms, and various other things. The X.509 standard is a crucial part of the stack functionality regarding TLS, but for the WGM110 Module integrator and for the scope of this application note, it is enough to manage the certificates using the BGScript/BGAPI commands.

The X.509 certificates can be stored in the WGM110 flash or RAM, in a location reserved for the keys. This is true except for the client certificates (private keys), which are stored only in the Module's RAM. If the certificates are part of the project build, they will be stored in the Flash, and their number is only limited by the amount of free flash. The build log from bgbuild/BGTool build software indicates the available flash space as shown below. For certificates loaded in run-time, if they are stored in Flash, the limit is 4 KB, and if they are stored in RAM, they are only limited by the amount of free RAM.



```
 in        :fw/wifi.juo

script
 compiler   :C:/Bluegiga/wgm110_0.y.0/bin/script_compiler.exe
 script     :main.bgs
 api        :C:/Bluegiga/wgm110_0.y.0/api/wifiapi.xml
 stack      :512
 variables  :358


CERT  :Equifax_Secure_Certificate_Authority.pem
  size:1160
BOOTLOADER:
 in        :C:/Bluegiga/wgm110_0.y.0/bin/../fw/boot.juo
uart:0
 location   :0
 bit rate   :115200
 handshake  :false
 BGAPI      :true
port:PA
 TRIS       :0
 LAT        :0
 ODC        :0
port:PB
 TRIS       :0
 LAT        :0
 ODC        :0
port:PC
 TRIS       :0
 LAT        :0
 ODC        :0
port:PD
 TRIS       :0
 LAT        :0
 ODC        :0
port:PE
 TRIS       :0
 LAT        :0
 ODC        :0
port:PF
 TRIS       :0
 LAT        :0
 ODC        :0

Memory used:
SW    :333304
HW    :168
USB   :0
Script:1908
Files :1
Certs :826
Free  :712369/1048576(68%)
ALL OK
```

Figure 2.1. Checking the Amount of Free Memory from the BGBuild Output

2.1 Storing an X.509 Certificate

A certificate can be entered into the Module's flash memory either during programming (compiled into the firmware) or during run-time. The certificates stored during programming need to be specified in the project file of the WGM110 firmware.

Specifying certificates during programming

Each certificate is listed in the project file separately, as shown in the example below.

Example: Specifying certificates in the project file

```
<certificates>
    <certificate path="certificate1.pem" format="pem"/>
    <certificate path="certificate2.dem" format="dem"/>
</certificates>
```

Specifying certicates during run-time

The certificate data can be sent from a host controller (e.g. MCU or PC) or read from an external memory chip using BGScript. For storing a certificate during run-time, 3 commands are required, as shown in the example below.

Example: Specifying certificates during run-time

```
call x509_add_certificate(0,1020)
call x509_add_certificate_data(255, certificate1(:))
call x509_add_certificate_data(255, certificate2(:))
call x509_add_certificate_data(255, certificate3(:))
call x509_add_certificate_data(255, certificate4(:))
call x509_add_certificate_finish()(result, fingerprint_len, fingerprint_data)
```

The command `x509_add_certificate_data` has a payload limit of 255 bytes, so typically several calls to this command are required to add the full certificate data into memory. The last command gives as a response a fingerprint, which can be regarded as a unique identifier between certificates, which can be used for run-time management, like deleting a particular certificate from the certificate store.

2.2 Listing the Certificate Store

The full list of certificates, which are stored in the Module (either in Flash or RAM), can be retrieved using the command `x509_list_certificates`. This command will generate the following two events for each certificate in the store:

- `x509_certificate_event` which carries the certificate fingerprint as a parameter
- `x509_certificate_subject` which carries the certificate subject as a parameter

When all the certificates in the store have been retrieved one final event `x509_certificates_listed` is generated to notify that the listing has ended.

2.3 Deleting an X.509 Certificate

Certificates might become obsolete, compromised, or the certificate store of the Module could run out of free space for new keys. In these cases, keys can be deleted using the `x509_delete_certificate` command, as shown in the example below.

Example: Deleting a certificate from the certification store

```
call x509_delete_certificate(fingerprint_len, fingerprint_data)
```

Note: Different certificates in the store are identified by the use of unique fingerprint numbers for each certificate.

2.4 Resetting the X.509 Certificate Store

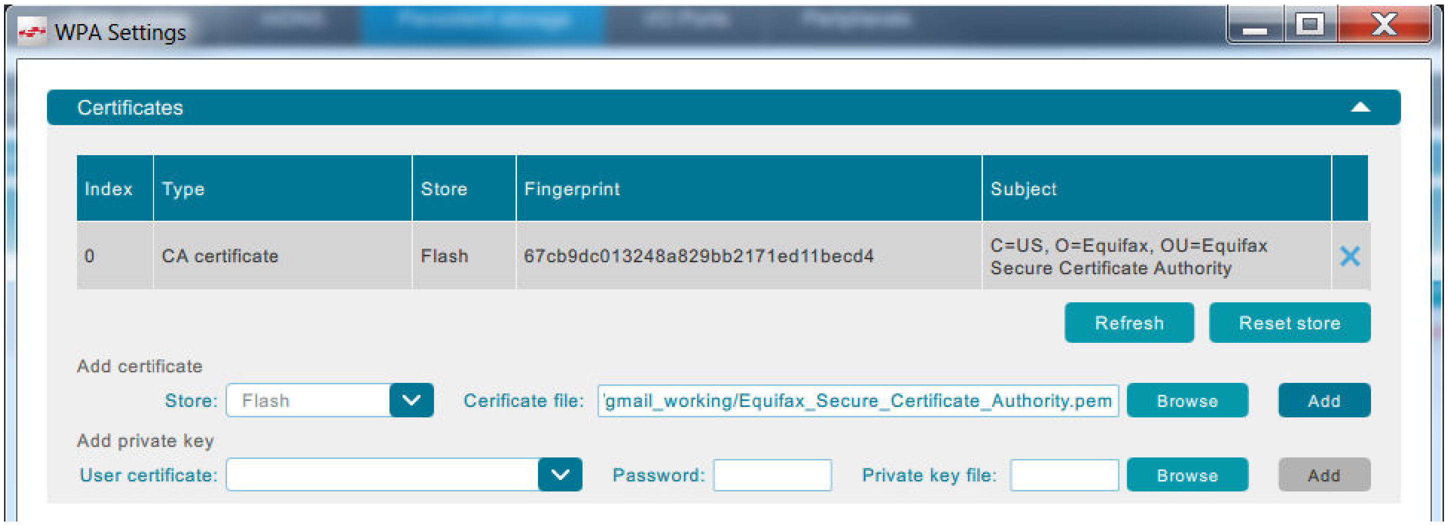
The command `x509_reset_store` will return the state of the certification store of the Module to the last flash reprogram state by erasing the run-time keys and by restoring the keys which were written during the build phase.

Example: Resetting the X.509 certificate ctore

```
call x509_reset_store()
```

2.5 Managing Certificates with BGTool

BGTool allows you to freely manage the certificates in the WGM110 Module, as explained in previous sections. Managing certificates with BGTool is illustrated in the screenshot below, which shows the certification store for the Gmail example. This example contains only one certificate to verify the *gmail smtp server*. In addition to listing the certificate store, BGTool also allows you to load the Flash or RAM memory or delete individual certificates. It is also possible to reset the entire certificate store. These functions are accessed by pressing the **WPA Settings** button in **Network -> "STA Mode"** after which a window, as the one shown in the screenshot below, will open.



Index	Type	Store	Fingerprint	Subject
0	CA certificate	Flash	67cb9dc013248a829bb2171ed11becd4	C=US, O=Equifax, OU=Equifax Secure Certificate Authority

Add certificate
Store: Flash
Cerificate file: 'gmail_working/Equifax_Secure_Certificate_Authority.pem

Add private key
User certificate:
Password:
Private key file:

Figure 2.2. Managing Certificates with BGTool

3. TLS

TLS is an acronym for *Transport Layer Security*, which is a security protocol succeeding the widely used and adopted SSL (*Secure Socket Layer*). SSL has become obsolete due to various vulnerabilities. The TLS protocol secures data between two TCP sockets (the client and the server) over the Internet. The WGM110 Module supports TLS versions 1.0, 1.1, and 1.2 in Client mode.

Each TLS connection requires significant resources from the WGM110 Module, so there can only be one TLS connection open at a time, but these can be concurrent with other TCP/UDP connections.

3.1 Setting TLS Authentication Mode

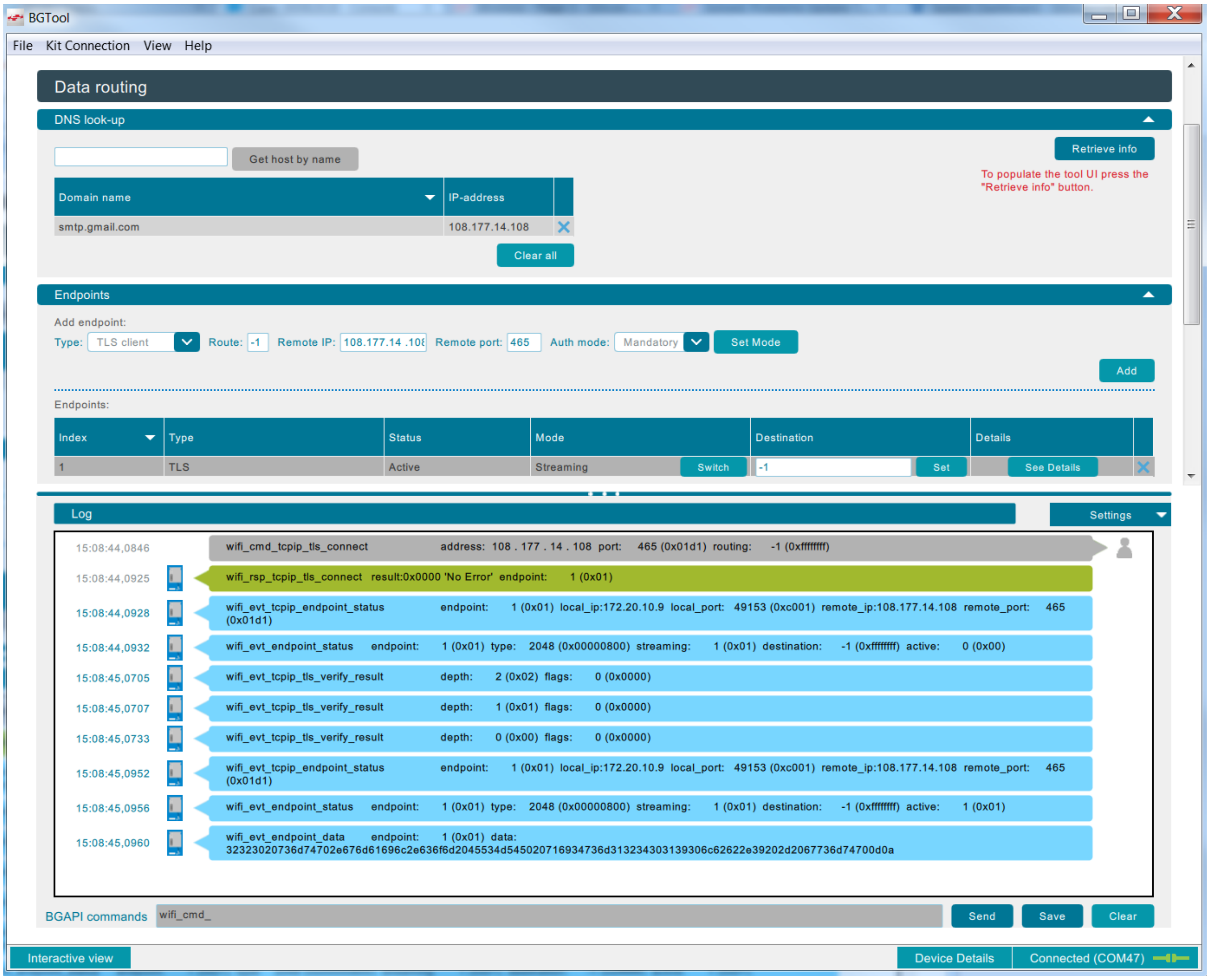
Establishing a TLS connection can be done with or without certificate verification, by using the command `tcpip_tls_set_authmode`. Verification will be disabled if the command parameter is *"0"*, in which case no certificate is required for establishing the connection. If the parameter is *"1"* , the certificate is verified, but a failed verification will not prevent the connection from being established, and, finally, if the parameter is *"2"* , the connection is only established if the certificate is correctly verified.

The default authentication mode is the one in which the connection is not established if the certificate verification fails (parameter *"2"*). The result of the validation is given through the event `tcpip_tls_verify_result`.

3.2 Opening and Closing a TLS Connection

Connections to a TLS server are managed in the same way as regular TCP connections. The command `tcpip_tls_connect` is used to connect to a server. The parameters given are the server's IP address, TLS port of the remote server, and the endpoint into which to route the incoming data from the server. If the endpoint is *"-1"* then all data will be routed to all BGAPI endpoints (e.g. UART in BGAPI mode, BGScript) and data will be received as `endpoint_data` events.

It is also possible to use BGTool to establish TLS connections, as shown in the next screenshot. The *smpt.gmail.com* IP address was resolved using the **DNS Look-up** functionality before establishing the connection. From the **Log** window it is possible to see that the certificates were correctly verified (result = *"0"*) and that the TLS connection was opened with endpoint index *"1"*. This is the endpoint number that must be used to send data to the server using the `endpoint_send` command or as a streaming destination from another endpoint, such as when the UART is in streaming mode.



Domain name	IP-address
smtp.gmail.com	108.177.14.108

Endpoints:

Index	Type	Status	Mode	Destination	Details
1	TLS	Active	Streaming	-1	

```
15:08:44,0846  wifi_cmd_tcpip_tls_connect        address:  108 . 177 . 14 . 108  port:   465 (0x01d1)  routing:   -1 (0xffffffff)
15:08:44,0925  wifi_rsp_tcpip_tls_connect  result:0x0000 'No Error'  endpoint:    1 (0x01)
15:08:44,0928  wifi_evt_tcpip_endpoint_status        endpoint:    1 (0x01)  local_ip:172.20.10.9  local_port:  49153 (0xc001)  remote_ip:108.177.14.108  remote_port:   465
               (0x01d1)
15:08:44,0932  wifi_evt_endpoint_status    endpoint:    1 (0x01)  type:   2048 (0x00000800)  streaming:    1 (0x01)  destination:    -1 (0xffffffff)  active:    0 (0x00)
15:08:45,0705  wifi_evt_tcpip_tls_verify_result        depth:    2 (0x02)  flags:    0 (0x0000)
15:08:45,0707  wifi_evt_tcpip_tls_verify_result        depth:    1 (0x01)  flags:    0 (0x0000)
15:08:45,0733  wifi_evt_tcpip_tls_verify_result        depth:    0 (0x00)  flags:    0 (0x0000)
15:08:45,0952  wifi_evt_tcpip_endpoint_status        endpoint:    1 (0x01)  local_ip:172.20.10.9  local_port:  49153 (0xc001)  remote_ip:108.177.14.108  remote_port:   465
               (0x01d1)
15:08:45,0956  wifi_evt_endpoint_status    endpoint:    1 (0x01)  type:   2048 (0x00000800)  streaming:    1 (0x01)  destination:    -1 (0xffffffff)  active:    1 (0x01)
15:08:45,0960  wifi_evt_endpoint_data      endpoint:    1 (0x01)  data:
               32323020736d74702e676d61696c2e636f6d2045534d5450207169347 36d313234303139306c62622e39202d2067736d74700d0a
```

Figure 3.1. Establishing a TLS Connection Using BGTool

4. Gmail Example Project

This example project first connects to an Access Point after power-on. Connection to an Access Point is indicated by **LED1** on the WSTK Main Board (see the next figure). When push button **PB0** is pressed, a TLS connection to a Gmail SMTP server is opened and an email with the message *"Hello World"* is sent to the server, this being indicated by **LED0**, which is on during the sending of the email. The placement of **LED0**, **LED1**, and **PB0** on the WSTK Main Board are indicated in the figure below.

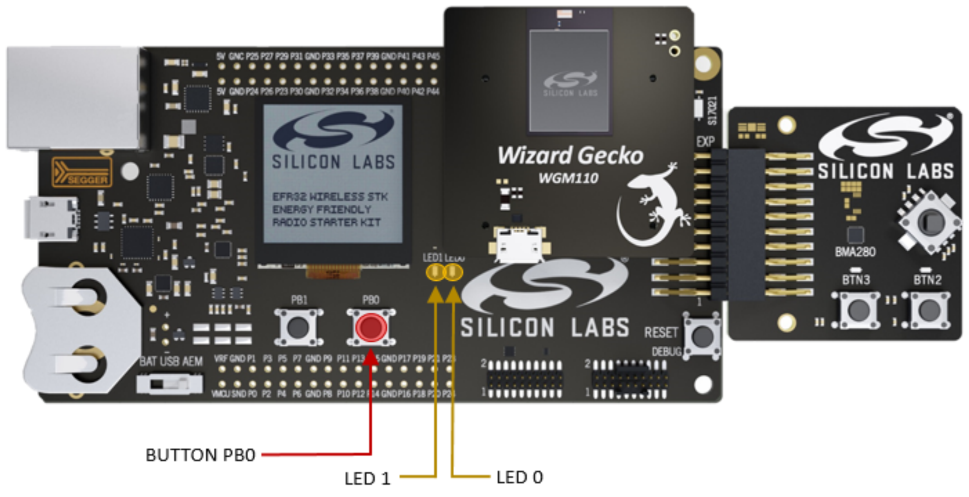


Figure 4.1. Location of LED0, LED1, and PB0 on the WSTK Main Board

A high level flow-chart of the project is shown below.

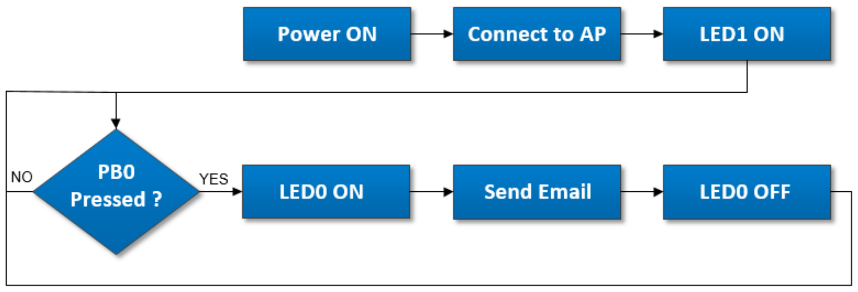


Figure 4.2. High-Level Flow Chart of the Gmail Example Project

This demo does not work "out of the box" because it requires a connection to the Internet via an Access Point (AP) and a Gmail account. The user needs to add the AP's SSID and password as well as the Gmail account's username and password (Base64 encoded). The Gmail account must be configured to enable "less secure apps" for the example SMTP connection to work. This is done by going to the account settings: "Sign-in & Security" and turning ON "Allow less secure apps". In addition, the user must download the root certificate required for the TLS connection.

Further instructions can be found in the *readme.txt* file inside the related project folder installed by the SDK. Relevant sections of the BGScript files are complemented by comments.

4.1 Project Configuration

Building a Wi-Fi project always starts by making a project file, which is a simple XML file defining all the resources used in the project. The project file of this example is shown in the example below. The table after the example lists each part of the project file with detailed descriptions of each tag.

Example: Project file contents example

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
   <scripting>
      <script in="main.bgs"/>
   </scripting>
   <hardware>
      <uart channel="0" baud="115200" api="true" handshake="false"/>
      <kit vcom="true"/>
   </hardware>
   <image out="gmail.bin" out_hex="gmail.hex"/>
   <certificates>
      <certificate path="Equifax_Secure_Certificate_Authority.pem" format="pem"/>
   </certificates>
</project>
```

Table 4.1. Project file contents description

Tags	Descriptions
`<project>`	This tag starts the project definition and the project file must end in </project> tag.

`<scripting>`
`<script in="main.bgs" />`
`</scripting>`
	The main BGScript code file is defined inside the <scripting> tags.

`<hardware>`
`<uart channel="0" baud="115200" api="true" handshake="false" />`
`<kit vcom="true" />`
`</hardware>`
	The <hardware> tag contains the device configuration and it can be embedded in the project definition or created as a separate file.
	
	In this project it defines the following:
	• The <uart> tag configures UART0 with baud-rate 115200, BGAPI enabled and handshake/flow-control disabled.
	• The <kit> tag enables the WSTK virtual serial communication.

`<image out="gmail.bin" out_hex="gmail.hex" />`	The <image> tag defines the name of the BGBuild compiler output files in both bin and hex formats. The generated files contain the Wi-Fi stack, hardware configuration and BGScript application they and can be loaded into the module's Flash.

`<certificates>`
`<certificate path=" Equifax_Secure_Certificate_Authority.pem" format="pem" />`
`</certificates>`
	The <certificates> tag lists all the certificates to be embedded in the flash at programming time. For this example only one certificate is required.

Note: The full syntax of the project configuration file and more examples can be found in the *UG161: WGM110 Wi-Fi® Module Configuration User's Guide*.

4.2 BGScript Walkthrough

This section explains the most relevant parts of the BGScript code used in this demo application and explains how the application works. The code explanations in this application note are categorized by the feature they apply to. This should help to understand which parts of the script implement which feature. References to the BGScript file are made in each image caption where relevant.

For more information on BGScript and its use, please refer to the *UG170: Wizard Gecko BGScript User's Guide*.

4.2.1 Connecting to an Access Point

The Access Points SSID and password are hardcoded into the BGScript file, as explained in the *readme.txt* file inside the project folder. The default mode of WGM110 Module is **Station Mode**, so in the `system_boot()` event, the Wi-Fi can be turned on using the command `sme_wifi_on()`. Once the Wi-Fi is powered on, the `sme_wifi_is_on()` event will be raised where the connection to the AP will be initiated as shown in the next figure.

Example: Connecting to an Access Point in *sta.bgs*

```
# Event received after Wi-Fi has been switched on.
event sme_wifi_is_on(state)
   # Set channel and passphrase then connect to known wireless network
   call sme_set_password(ap_psk_len, ap_psk(:))
   call sme_connect_ssid(ap_ssid_len, ap_ssid(:))
end
```

Once the connection is established and the TCP/IP stack is ready the `sme_interface_status()` event will be raised and **LED1** is then turned **ON** informing the user that connection has been established.

The related BGScript code is listed in the example below.

Example: Access Point connection "Up" indications with LED1 of WSTK

```
# Event reporting the status of the wireless network interface
event sme_interface_status(hw_interface, interface_status)
   if interface_status = 0 then
      status = 99
      call hardware_write_gpio(GPIO_PORTC, $0002, $0000)
   else # When connection to Wi-Fi network is up and TCP/IP stack is ready
      status = 0
      call hardware_write_gpio(GPIO_PORTC, $0002, $0002) # Turn on WSTK LED1
   end if
end
```

Once the connection is up, the WGM110 Module will be waiting for a press on push button **PB0** to initiate the establishment of a TLS connection to the Gmail SMTP server and to send the email.

4.2.2 Resolving Host and Opening TLS Connection

A press on the push button **PB0** is catched by a `hardware_interrupt()` event in BGScript. When the event is catched, the application starts by checking that it is in the correct status to initiate the TLS connection. If the status is correct, it will turn on **LED0** to inform the user that the process has started and it will continue to resolve the host name in order to retrieve the IP for *smtp.gmail.com* as shown in the next example.

Example: Resolving the host name and opening a TLS connection in *main.bgs*

```
event hardware interrupt(interrupts, timestamp)
   if (interrupts & 4) && (status = 0) then
      status = 1
      call hardware_write_gpio(GPIO_PORTC, $0001, $0001) # Turn on WSTK LED0
      call tcpip_dns_gethostbyname(gmail_smtp_server_name_len, gmail_smtp_server_name(:))
   end if
end

event tcpip_dns_gethostbyname_result(dns_result, server_ip_address, name_len, name_data)
   call tcpip_tls_set_authmode(2)
   call tcpip_tls_connect(server_ip_address, gmail_smtp_server_ssl_port, -1)(result, endpoint_gmail_smtp_server
_ssl)
end
```

When the host name has been resolved, the `tcpip_dns_gethostbyname_result()` event is raised. The verification mode is set to mandatory, which means that if the certificate verification fails the connection will not be established.

The command `tcpip_tls_connect()` is then called to establish the TLS connection with the server. It takes as its parameters the TLS server IP address (retrieved by resolving the *smtp.gmail.com* host name), the port number (which in case of Gmail SMTP server is 465), and the endpoint into which to forward incoming data (*"-1"* indicates that data should be exposed as BGAPI `endpoint_data()` events). The command then returns the result (*"0"* if successful) and the endpoint to which outgoing data should be sent, using the `endpoint_send()` command.

The correct certificate for the TLS connection is automatically retrieved from the certificate store, and it is verified before establishing the connection.

4.2.3 Sending Email with SMTP Protocol

The SMTP protocol is not natively supported by WGM110 Module via API, but for simple use cases, such as the one in this demo, it is possible to implement the message exchanges with the SMTP server using BGScript. Because the BGScript code required for this exchange is lengthy, it will not be shown in this document. Please refer to the *main.bgs* BGScript file and the `endpoint_data()` events for details.

Example: SMTP data exchange

```
S: 220                               // Hello, server is ready
C: EHLO                              // Greet the server
S: 250                               // Action OK
C: AUTH LOGIN                        // Request to authenticate client
S: 334                               // Request accepted, send username and password in base64 encoded string
C: username                          // Username
S: 334                               // Accepted
C: password                          // Password
S: 235                               // Authentication succeeded
C: MAIL FROM:<someone@gmail.com>     // Sender's email address
S: 250                               // Action ok
C: RCPT TO:<someone@somewhere.com    // Recipient's email address
S: 250                               // Action ok
C: DATA                              // Beginning of message text
S: 354                               // Start mail input. End data with <CR><LF>, <CR><LF>
C: From: someone@gmailc.com
C: To: someone@somewhere.com
C: Subject: Hello World
C:
C: Hello world, this is WGM110.
C:
C: .                                 // <CR><LF>.>CR><LF> end-of data sequence
S: 250                               // Action OK
C: QUIT
S: 221                               // Bye
(The server closes the connection)
```

5. Revision History

5.1 Revision 1.2

May 23rd, 2016

Changes: Code examples redone, figures updated, sections 4.3 now 4.2.1, 4.4 now 4.2.2 and 4.5 now 4.2.3.

5.2 Revision 1.1

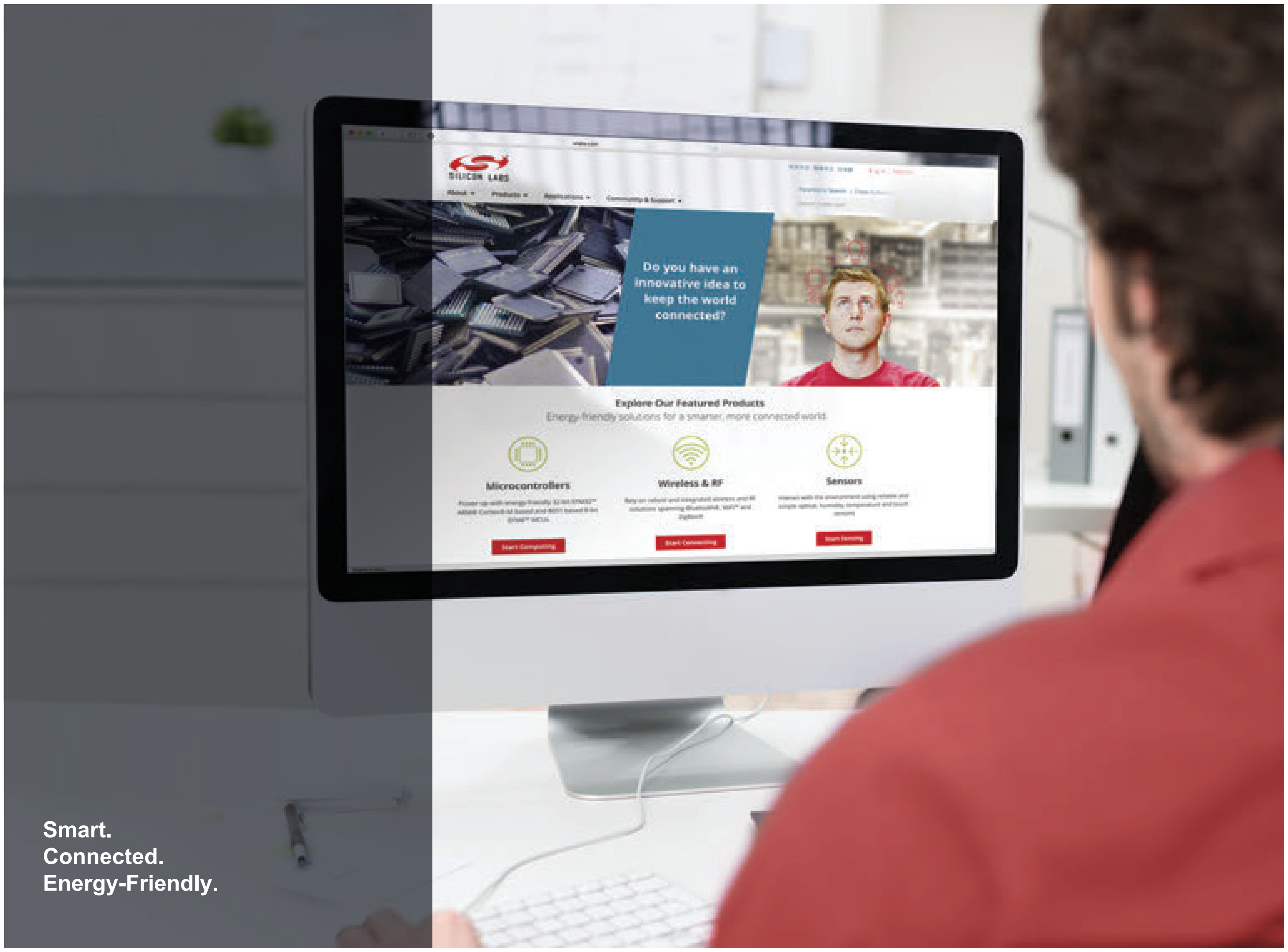
March 15th, 2016

Updated Section 4: Gmail Example Project

5.3 Revision 1.0

February 22nd, 2016

Initial release.


Smart.
Connected.
Energy-Friendly.
Do you have an innovative idea to keep the world connected?
Explore Our Featured Products
Microcontrollers
Wireless & RF
Sensors

Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com