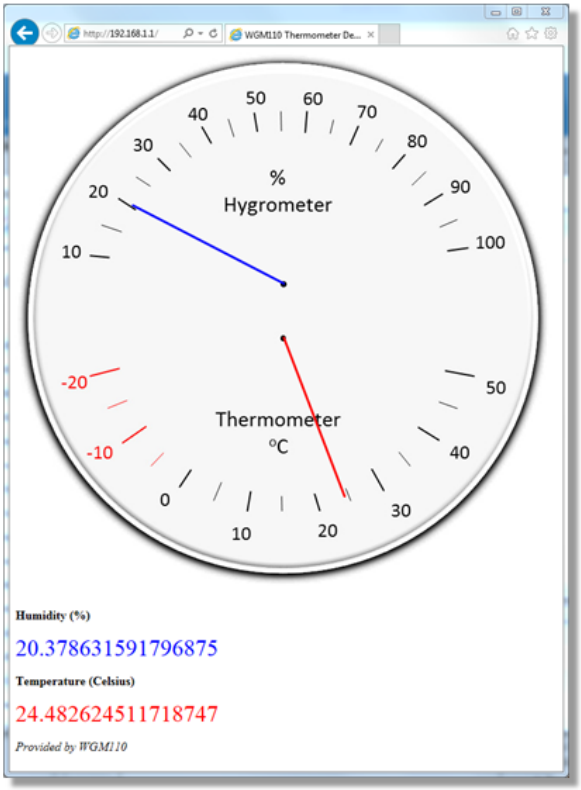
AN967: Wizard Gecko WSTK Demo Walkthrough

This document describes the sensor example project, which measures temperature and humidity and displays the values on a standard web browser. This example project is included in the WGM110 SDK and is the factory demo programmed into the WGM110 Module located on the BRD4300A Radio Board of the Wizard Gecko Module Wireless Starter Kit (SLWSTK6120A).

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
  - BGScript usage
  - HTTP Server
  - Wi-Fi Access Point
  - mDNS
  - Peripheral sensor usage
  - Timer usage



1. Walkthrough of the WGM110 WSTK Sensor Demo Application

This section walks you through the demo application that is pre-programmed into the WGM110 Modules delivered as part of the Wizard Gecko Wireless Starter Kits (WSTK) when they are shipped from the factory. This demo implements a webpage which displays the temperature and relative humidity values measured through the Silicon Labs Si7021 sensor located on the WSTK Main Board. The demo is implemented with the BGScript scripting language and runs fully inside the WGM110 Module with no need for an external host. The application also demonstrates basic peripheral connectivity as it uses the I^2C interface to communicate with the Si7021 sensor chip.

The basic architecture of an application implemented with BGScript for stand-alone operation is shown in the figure below. In addition to stand-alone and NCP mode (network co-processor), the WGM110 Module also supports mixed-mode architecture where some tasks can be executed via BGScript while others are controlled via a host.

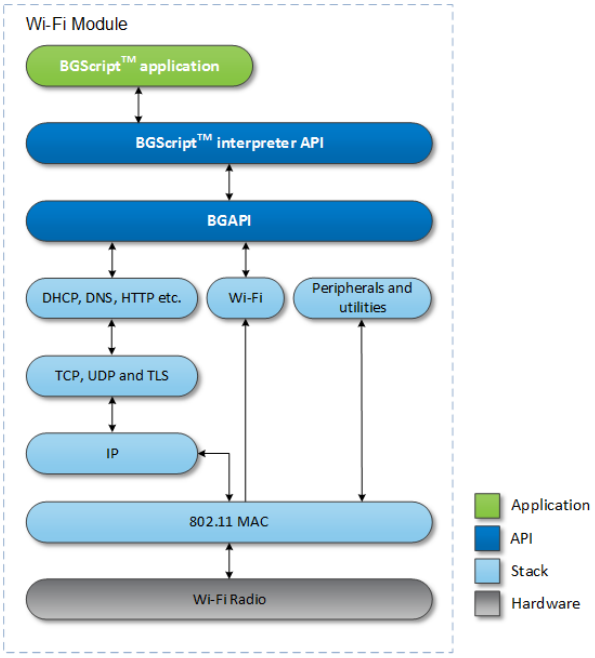


Figure 1.1. WGM110 Stand-Alone Application Architecture

The Sensor example project is divided into three main groups, as illustrated in the figure below. These are the **Project Configuration** (*project.xml* file), the **HTTP Server** (*index.html* and *meteringUI.jpg* files), and the **BGScript Application** (*main.bgs* together with *ap.bgs*, *api.bgs*, *i2c.bgs* and *mdns.bgs* files) groups.

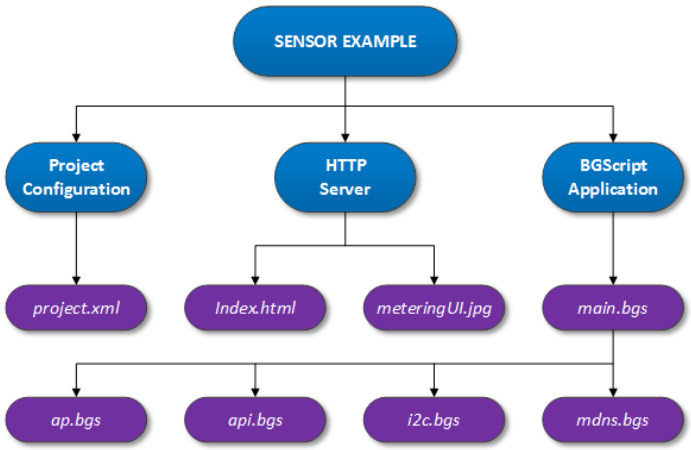


Figure 1.2. Sensor Project Structure

1.1 Project Configuration

Building a Wi-Fi project always starts by making a Project file which is a simple XML file that defines all the resources used in the project. An example of a typical Project file is listed in the figure below. The example's content is explained in the following table.

Example: Project file contents

```
<project>
   <scripting>
       <script in="main.bgs"/>
   </scripting>
   <hardware>
       <uart channel="0" baud="115200" api="true" handshake="false"/>
       <kit vcom="true" sensor="true"/>
       <i2c channel="1" location="2"/>
   </hardware>
   <image out="sensor.bin" out_hex="sensor.hex"/>
   <files>
       <file path="index.html"/>
       <file path="meteringUI.jpg"/>
   </files>
</project>
```

The table below indicates explanations to the code example above.

Table 1.1. Project File Example Explained

XML Attributes/Attribute pairs	Description
`<project>`	This attribute starts the project definition file.

`<scripting>`
`<script in="main.bgs" />`
`</scripting>`
	The main BGScript code file is defined within the `<scripting>` and `</scripting>` attibutes.

`<hardware>`
`<uart channel="0" baud="115200" api="true" handshake="false" />`
`<kit vcom="true" sensor="true" />`
`<i2c channel="1" location="2" />`
`</hardware>`
	The device configuration is defined within the `<hardware>` and `</hardware>` attributes.
	Note: Device configuration can be embedded in the project definition or created as a separate file.
	In this project it defines:
	• The `<uart>` attribute configures **UART0** with baud rate 115200, BGAPI enabled and handshake/flow control disabled.
	• The `<kit>` attribute enables the WSTK virtual serial communication and I^2C bus routing to the SI7021 sensor chip.
	• The `<i2c>` attribute configures **I2C1** to be available in **Location 2**

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

`<files>`
`<file path="index.html" />`
`<file path="meteringUI.jpg" />`
`</files>`
	All the files to be embedded into the Flash memory of the Module are defined within the `<files>` and `</files>` attributes.
	These files are included in the *hex/bin* files generated by BGBuild or BGTool compiler.

`</project>`	This attribute ends the project definition file.

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

1.2 BGScript Code Walkthrough

This section explains the most relevant parts of the BGScript code used in the demo application and also explains how the application works. The code explanations in this application note are categorized by the feature to which they apply. 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 please refer to th UG170: Wizard Gecko BGScript User's Guide.

1.2.1 System Boot Event

The `system_boot` event is always generated when power is applied to the Module or a reset occurs, and it is the starting point for the code execution. Typically this event is handled in the main BGScript file.

Example: System Boot Event Handler in *main.bgs* File.

```
# Event received when the system has been successfully started up.
event system_boot(hw, bootloader_version, major, minor, build, revision_len, revision_data)

# Initialize modules
    call ap_mode_init()
    call api_init()
    call mdns_init()

# Set the correct operating mode.
    call sme_set_operating_mode(MODE_AP)

# Enable Wi-Fi, this call will trigger sme_wifi_is_on() event.
    call sme_wifi_on()
end
```

This event handler contains calls for initialization procedures located in other BGScript source files. These procedures initialize configuration parameters for AP Mode (channel, SSID, security) and mDNS services. The default operating mode of the WGM110 Module is **"Station Mode"**, and the correct mode must be set before turning on the Wi-Fi, as is done in this event handler.

1.2.2 Setting Up the Servers and Services

Once the command `sme_wifi_on` has been issued, there will be an event `sme_wifi_is_on` confirming that the Wi-Fi has been turned on. When this event is received, the `ap_mode_start` procedure is called, in which the values for the AP's TCP/IP settings are configured (IP address, netmask and gateway) and the AP Mode is started by calling `sme_start_ap_mode`. The AP name is "WGM110 Example," and it is set up without any security settings to allow easier demonstration.

Example: AP Mode Start-up in *ap.bgs* File

```
# Procedure for starting AP mode
export procedure ap_mode_start()
  # Set a static IP address (192.168.1.1)
     call tcpip_configure(192.168.1.1, 255.255.255.0, 192.168.1.1, 0)

  # Disable all power save functionality
     call system_set_max_power_saving_state(0)

  # Set the passphrase
     call sme_set_ap_password(ap_passphrase_len, ap_passphrase(0:ap:passphrase_len))

  # Start the AP mode. This call will trigger sme_ap_mode_started() event on success
  # and sme_ap_mode_failes on failure
     call sme_start_ap_mode(ap_channel, ap_security, ap_ssid_len, ap_ssid(0:ap_ssid_len))
end

# Event received after AP mode has been started
event smp_ap_mode_started(hw_interface)
  # Configure HTTP server paths
     call https_add_path(AP_PATH_DEVICE_API, AP_PATH_API_LEN, AP_PATH_API(:))
     call https_add_path(AP_PATH_DEVICE_FLASH, AP_PATH_ROOT_LEN, AP_PATH_ROOT(:))

  # Enable/disable HTTP/DHCP/DNS server
     call https_enable(ap_enable_https, ap_enable_dhcps, ap_enable_dnss)
end
```

This command generates the event `sme_ap_mode_started` where the HTTP Server resources are mapped and the server is enabled together with the DHCP and DNS servers. When the Wi-Fi interface is fully ready, the event `sme_interface_status` is generated, where the *main.bgs* procedure `on_ap_mode_started` is called. This procedure starts a soft timer used to trigger the sensor readings, sets pad **PC0** of the WGM110 Module to function as an output with logic level "high" to turn on **LED0** on the WSTK Main Board, and also starts the mDNS service.

Example: Starting the AP Mode in *main.bgs* File

```
# Procedure called when AP mode has been started
export procedure on_ap_mode_started()
   #Set a periodic timer to read current temperature
   call hardware_set_soft_timer(MEASUREMENT_INTERVAL, TIMER_MEASUREMENT, 0)

   # Config IO port direction - Set port C pin 0 as output (LED)
   call hardware_configure_gpio(GPIO_PORTC, 0, GPIO_MODE_OUTPUT, GPIO_OUTPUT_LOW)
   # Set PC0 to indicate AP mode has been started
   call hardware_write_gpio(GPIO_PORTC, $0001, $0001)

   # Start mDNS
   call mdns_start()
end
```

1.2.3 Reading the Sensor Humidity and Temperature Data

The Si7021 is a relative humidity and temperature sensor. It is accessible through an I^2C interface. The soft timer was configured earlier to generate an event every 1000 milliseconds. When the event `hardware_soft_timer` occurs the procedure `i2c_measurement` is called which handles the actual sensor data acquisition.

Example: Reading of Si7021 Sensor Data in *i2c.bgs* file

```
# Procedure for reading current Temperature and Humidity measurements from Si7021
export procerude i2c_measurement()

   # Humidity measurement :

   # Write 1 byte to slave address 0x40
   call i2c_start_write(I2C_SLAVE_CHANNEL, I2C_SLAVE_ADDRESS, 1, Measure_CMD_Humidity)
   # Write complete
   call i2c_stop(I2C_SLAVE_CHANNEL)
   # Read from 2 byte slave address 0x40
   call i2c_start_read(i2C_SLAVE_CHANNEL, I2C_SLAVE_ADDRESS, 2)(cmd_result, cmd_value_len, cmd_value(0:cmd_valu
e_len))
   # Read complete
   call i2c_stop(I2C_SLAVE_CHANNEL)
   # Update Humidity
   i2c_humidity = (cmd_value(0:1) << 8) | cmd_value(1:1)

   # Temperature measurement :

   # Write 1 byte to slave address 0x40
   call i2c_start_write(I2C_SLAVE_CHANNEL, I2C_SLAVE_ADDRESS, 1, Measure_CMD_Temperature)
   # Write complete
   call i2c_stop(I2C_SLAVE_CHANNEL)
   # Read from 2 byte slave address 0x40
   call i2c_start_read(i2C_SLAVE_CHANNEL, I2C_SLAVE_ADDRESS, 2)(cmd_result, cmd_value_len, cmd_value(0:cmd_valu
e_len))
   # Read complete
   call i2c_stop(I2C_SLAVE_CHANNEL)
   # Update Temperature
   i2c_temperature = (cmd_value(0:1) << 8) | cmd_value(1:1)
end
```

For more details on how to read the relative humidity and temperature data from the sensor please refer to the *Si7021 Data Sheet*.

1.2.4 mDNS/DNS-SD

The mDNS lists two services, the “WGM110 Sensor Page” and “WGM110 Sensor Service”. They are configured in the `mdns_init` procedure called from `system_boot` and started after the Wi-Fi interface has been set up successfully.

Example: Configuring and Starting Up mDNS in *mdn.bgs* File

```
# Procedure for initialising mDNS functionality
export procedure mdns_init()
   # Configure mDNS hostname
   call tcpip_mdns_set_hostname(6, "WGM110")

   # Configure DNS-SD services
   call tcpip_dnssd_add_service(80, 0, 4, "http")(cmd_result, mdns_service_index)
   call tcpip_dnssd_add_service_instance(mdns_service_index, 18, "WGM110 Sensor Page")
   call tcpip_dnssd_add_service(80, 0, 6, "sensor")(cmd_result, mdns_service_index)
   call tcpip_dnssd_add_service_instance(mdns_service_index, 21, "WGM110 Sensor Service")
   call tcpip_dnssd_add_service_attribute(mdns_service_index, 21, "path=/api/measurement")
   call tcpip_dnssd_add_service_attribute(mdsn_service_index, 11, "format=json")
end

# Procedure for starting mDNS
export procedure mdns_start()
   # Start mDNS service. This call will trigger tcpip_mdns_started() event on success
   # and tcpip_mdns_failed() on failure.
   call tcpip_mdns_start()
end

# Event called when mDNS service has been started.
event tcpip_mdns_start()
   # Start DNS-SD services
   call tcpip_dnssd_start_service(0)
   call tcpip_dnssd_start_service(1)
end
```

The mDNS services can be discovered, for example, using a mobile application, such as the Discovery – Bonjour Browser for iOS, where in which the Sensor service will show the attributes.

Figure 1.3. Screenshot with mDNS Service Discovery on Mobile Phone (iOS)

The service advertises a sensor service that can be used to directly fetch the sensor data in JSON format. This data can be accessed by typing `192.168.1.1/api/measurement` in a browser, which displays the raw JSON formatted data as shown in the next figure.

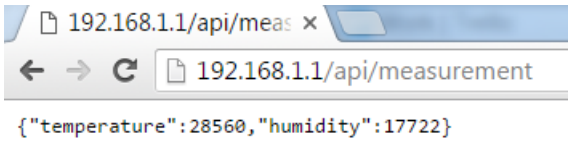


Figure 1.4. Sensor Data Read Through mDNS Service

1.2.5 HTTP Server

The sensor example contains an embedded web page which displays the sensor data readings. When the user opens a web browser and types the WGM110 IP address `192.168.1.1` into the address field of the browser, the browser will send a GET request for '/' resource as depicted in the wireshark log snippet below.

No.	Time	Source	Destination	Protocol	Length	Info
57	4.674181000	192.168.1.2	192.168.1.1	HTTP	431	GET / HTTP/1.1

Figure 1.5. Initial GET Request from Browser

The '/' path in the HTTP Server has been mapped to flash, and when this resource is requested the server will automatically respond by sending the `index.html` file. This is the default behavior of the HTTP Server when the path mapped to flash is requested, and it cannot be modified.

`index.html` contains html code to render the webpage page as well as javascript code. The javascript function `readMeasurement` is triggered once per second to send GET requests for the resource '/api/measurement' and parse the received JSON. The figure below shows a wireshark log snippet of one of the GET requests for the '/api/measurement' resource.

Example: Javascript code in *index.html* file for reading the sensor data from WGM110 Module

```
/**
 * Function for requesting the current measurement
 */
 function readMeasurement() {
     var xhr = new XMLHttpRequest();
     xhr.onreadystatechange = function() {
         if (xhr.readyState === 4 && xhr.status === 200) {
             myMeasurementsObj = JSON.parse(xhr.responseText);
             var temp = getTemperatureValue();
             var humi = getHumidityValue();
             // Draw the loaded image onto the canvas
             ctx.drawImage(img, 0 ,0);
             drawThermometer(temp);
             drawHygrometer(humi);
             setTimeout(readMeasurement, 1000);
         }
     };
     xhr.open("GET, "/api/measurement", true);
     xhr.send();
}
```

No.	Time	Source	Destination	Protocol	Length	Info
230	5.535575000	192.168.1.2	192.168.1.1	HTTP	375	GET /api/measurement HTTP/1.1

Figure 1.6. GET Request for Sensor Data

The '/api' path has been mapped to BGScript which means that all requests starting with '/api' will generate `https_api_request` events and they must be handled and responded via BGScript using `https_api_response` and `http_api_response_finish` commands (the latter one is issued only when all the response data has been sent). In this example, the BGScript code builds a JSON formatted response containing the sensor data which is sent as the response to the HTTP request.

Example: HTTP Response with the Sensor Data in *api.bgs* File

```
# Procedure for handling REST API measurement request
procedure api_handle_measurement_request(request)
   # Generate a response
   api_resp_len = 0

   # HTTP headers
   call api_add_http_headers()
   # JSON status object start
   call api_add_json_object_start()
   # Temperature name
   call api_add_json_object_name(API_PARAM_JSON_TEMPERATURE_LEN, API_PARAM_JSON_TEMPERATURE(:))
   # Temperature value
   call api_add_json_integer(i2c_temperature)
```

```
   # Adding a comma between JSON object element
   call api_add_comma()
   # Humidity name
   call api_add_json_object_name(API_PARAM_JSON_HUMIDITY_LEN, API_PARAM_JSON_HUMIDITY(:))
   # Humidity value
   call api_add_json_integer(i2c_humidity)
   # JSON status object end
   call api_add_json_object_end()

   # Send the response
   call https_api_response(request, api_resp_len, api_resp(0:api_resp_len))
   call https_api_response_finish(request)

   # Mark the request as handled
   api_req_handled = 1
end
```

The wireshark snippet in the figure below shows the response. The highlighted payload section is the HTTP header followed by the temperature and humidity JSON objects.

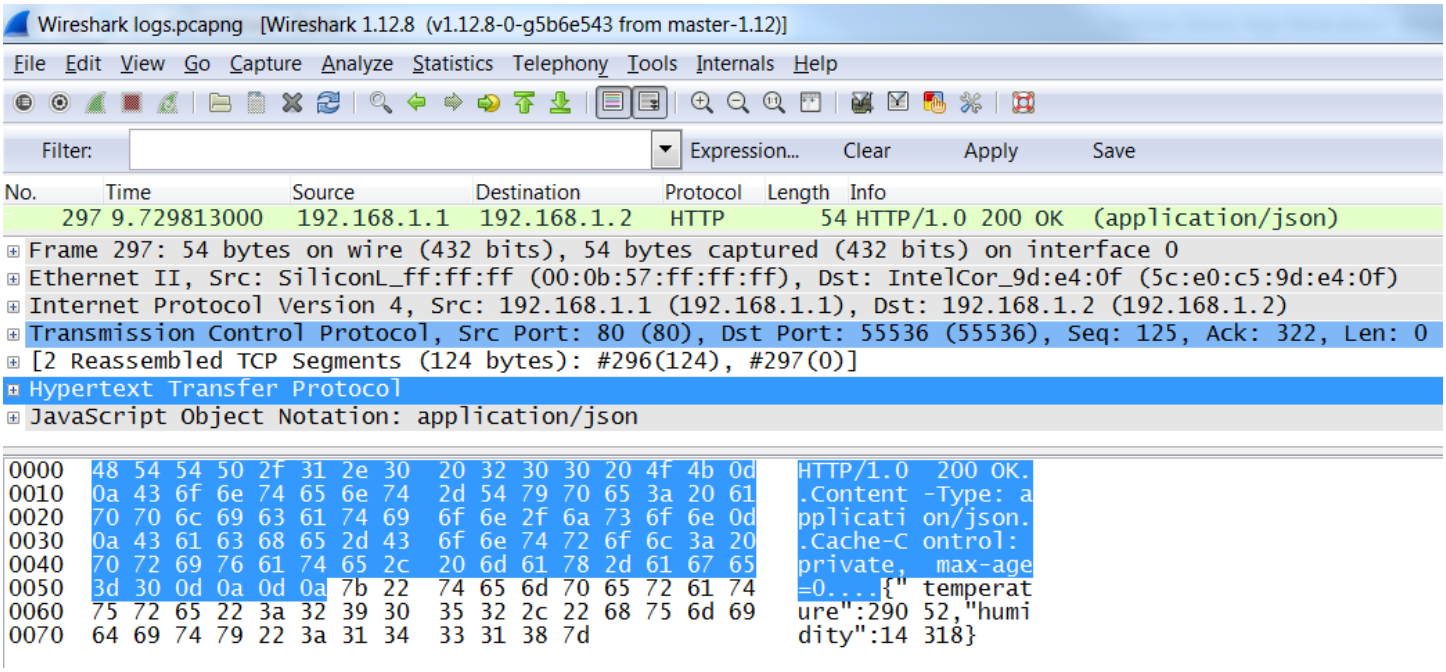


Figure 1.7. HTTP Response with JSON Formatted Data

The next figure contains a flow chart summarizing the exchanges occurring between the browser, the Wi-Fi stack, and BGScript. The starting point for the flow chart is the moment the user opens a browser and types the WGM110 IP 192.168.1.1 address in the address field of the browser.

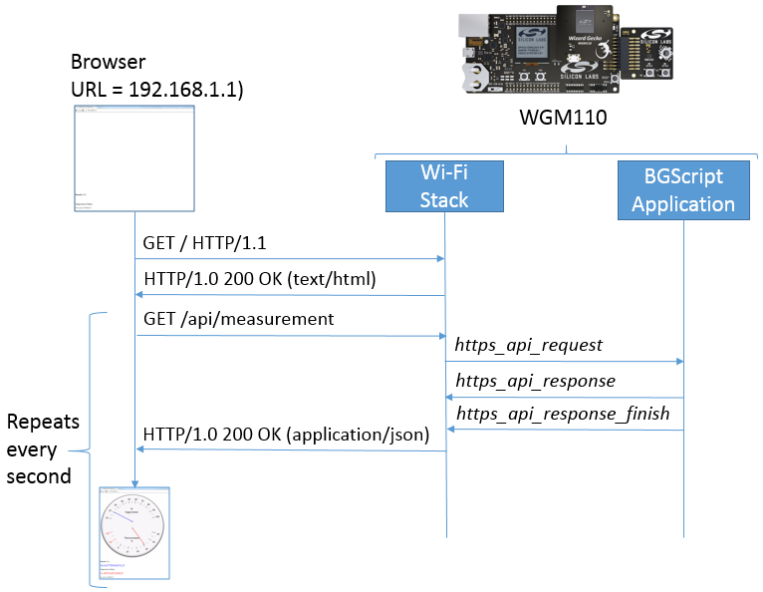


Figure 1.8. Flow Chart of Events to Read the Sensor Data using WGM110 Module

2. Revision History

2.1 Revision 1.1

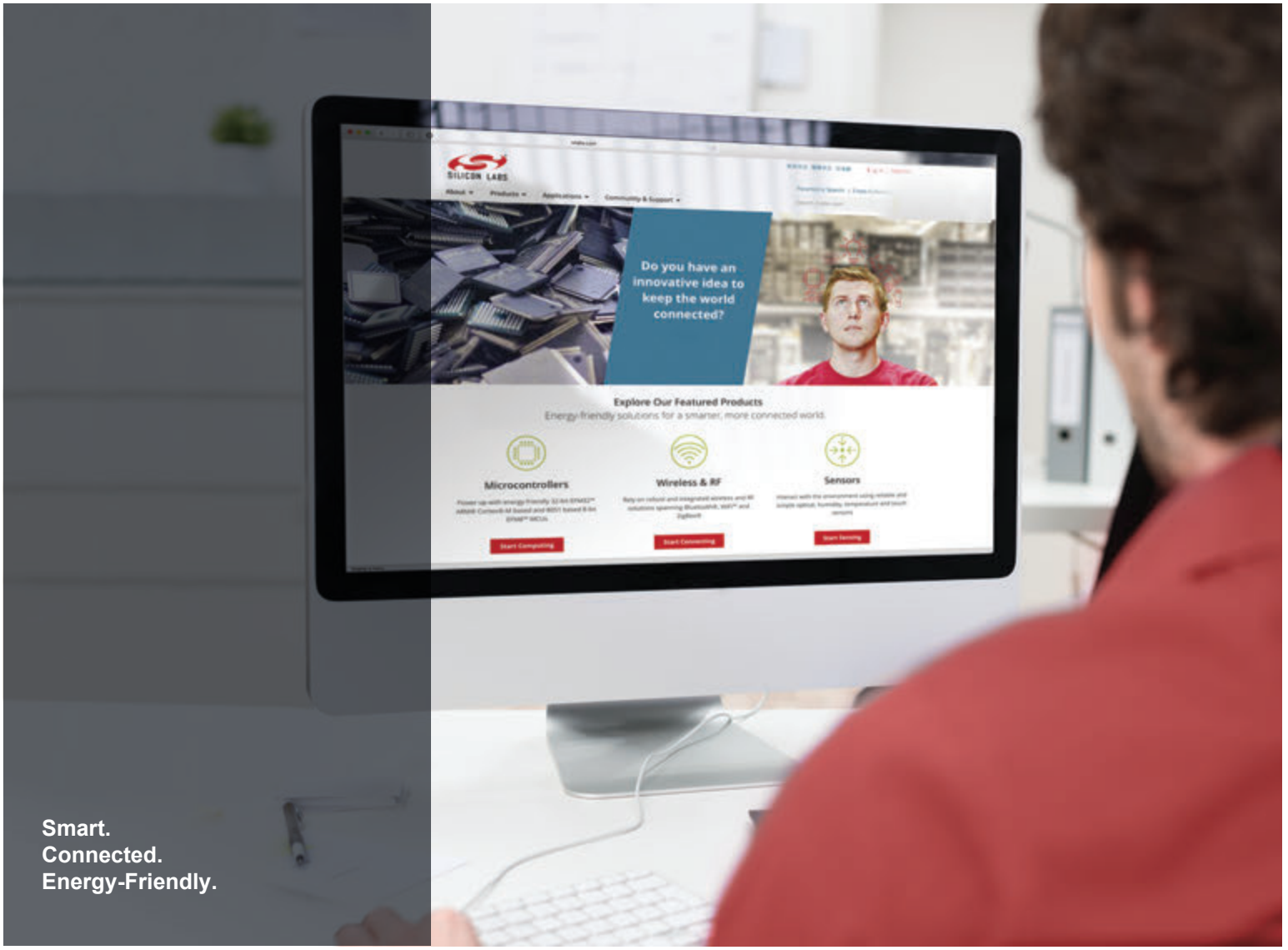
May 23rd, 2016

Code examples redone using codeblocks, figures updated.

2.2 Revision 1.0

February 22nd, 2016

Initial release.



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