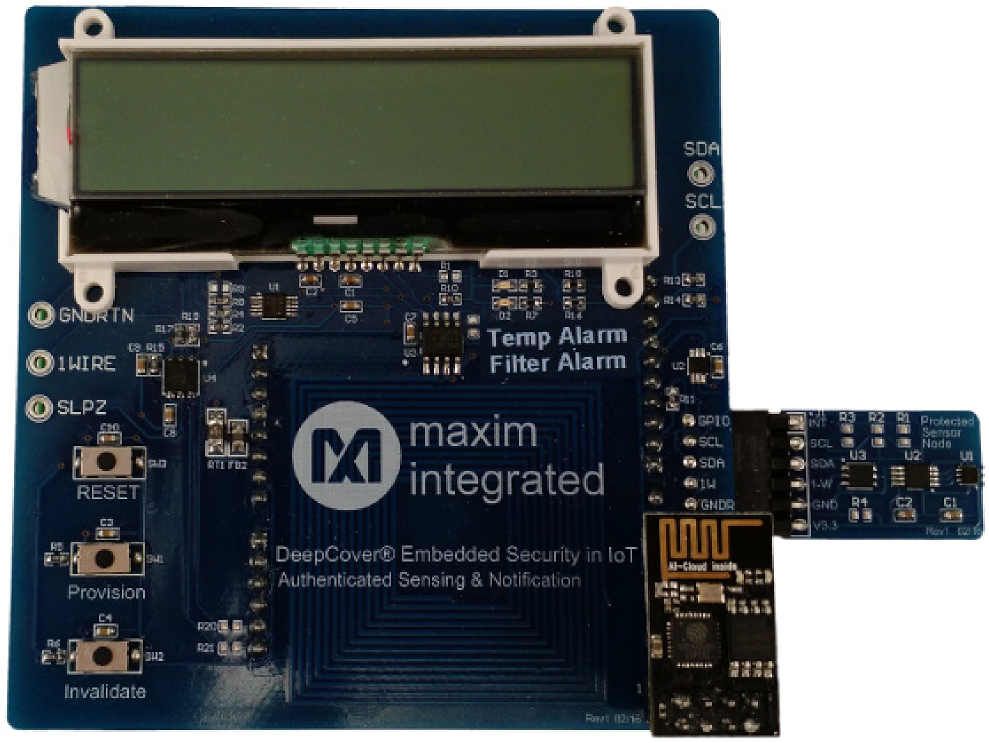


MAXREFDES143#
Quick Start Guide

UG6296; Rev 0; 6/16

DeepCover® Embedded Security in IoT:
Authenticated Sensing and Notification

Abstract

This Quick Start Guide provides information about preparing and running the MAXREFDES143# example application. The MAXREFDES143# subsystem reference design secures industrial IoT systems with hardware security provided by a DS2465 SHA-256 coprocessor and DS28E15 SHA-256 authenticator.

Table of Contents

Equipment

The example web application for this design is open source and can be run on any PHP-enabled web server. It is possible to create either a private instance of the web application with a personal server or utilize a shared instance provided by Maxim. A PHP-enabled web server and complete web server setup are only necessary to create a private instance of the web application. Maxim makes no guarantees regarding reliability, availability, or data security when utilizing the shared web application instance.

Required

MAXREFDES143# kit including shield, protected sensor node module, and ESP8266 module

MAX32600MBED# base board

Internet-connected computer with USB to load firmware

Optional

PHP-enabled web server

Configure Web Server (Optional)

The example web application uses PHP for server-side processing. PHP must be installed and configured to operate with a web server such as Apache. If you are new to web development and running Microsoft Windows®, consider using the tools developed by the open-source WampServer project. Visit the PHP Manual for more information about requirements for your system.

1. Download the latest web application source package from the MAXREFDES143# page on the Maxim website under the **Design Resources** tab.

2. Extract the package to the root of the web server hosting directory and set permissions as necessary.

3. Take note of the address and port for the web server. The user page should now be visible from the web browser (**Figure 1**).

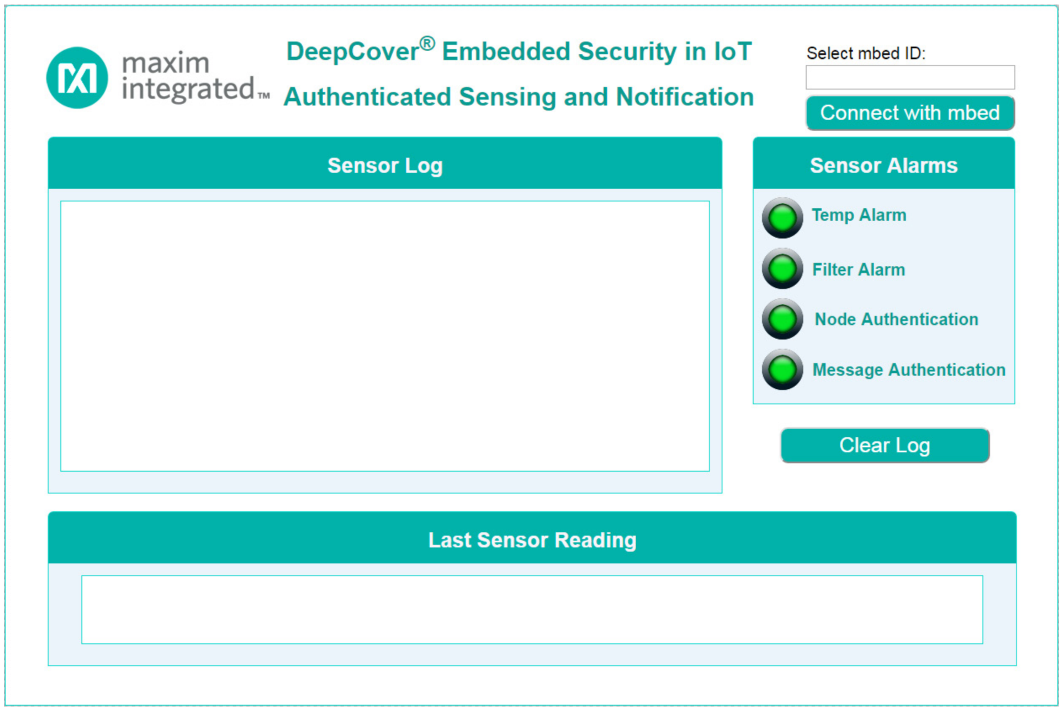
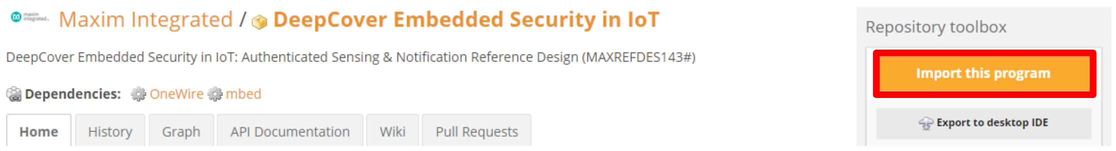


Figure 1. User page.

Configure Firmware

1. Load the mbed® repository page for the DeepCover® Embedded Security in IoT firmware program in a web browser.

2. Import latest revision of program into the online compiler by clicking on the **Import this program** button in the mbed repository page (**Figure 2**). A free mbed account is required to access the online compiler.



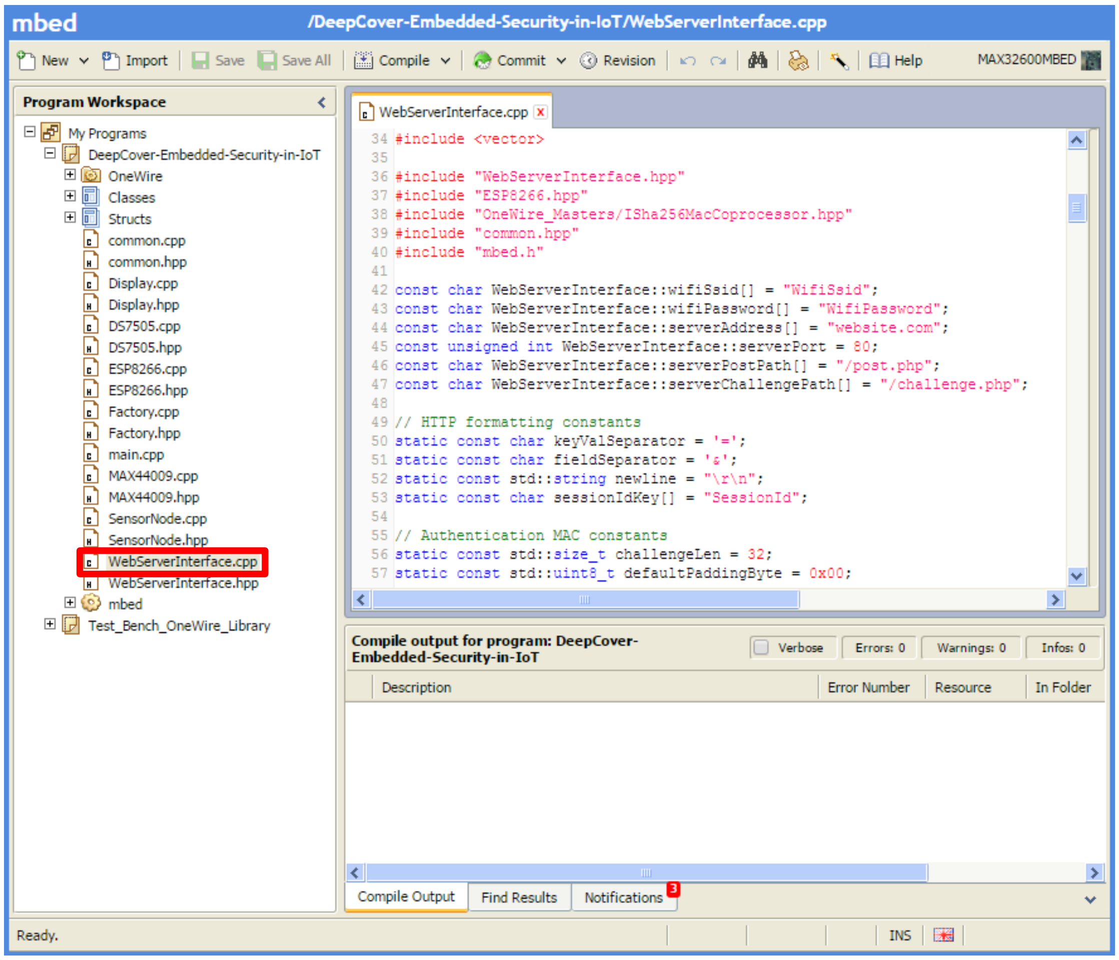
Maxim Integrated / DeepCover Embedded Security in IoT

DeepCover Embedded Security in IoT: Authenticated Sensing & Notification Reference Design (MAXREFDES143#)

Dependencies: OneWire mbed

Figure 2. "Import this program" button.

3. Open the source file **WebServerInterface.cpp** in the imported copy of **DeepCover-Embedded-Security-in-IoT** (Figure 3).



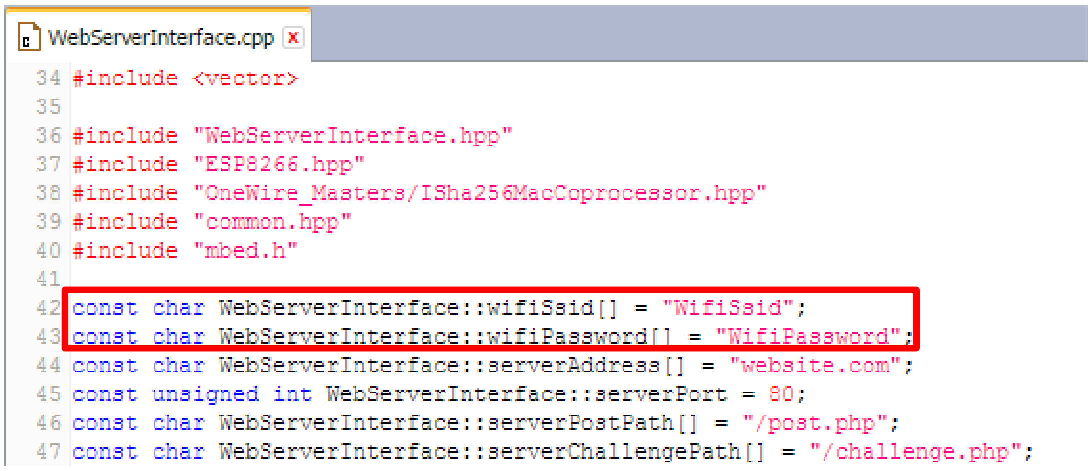
mbed /DeepCover-Embedded-Security-in-IoT/WebServerInterface.cpp

```
#include <vector>

#include "WebServerInterface.hpp"
#include "ESP8266.hpp"
#include "OneWire_Masters/ISha256MacCoprocessor.hpp"
#include "common.hpp"
#include "mbed.h"

const char WebServerInterface::wifiSsid[] = "WifiSsid";
const char WebServerInterface::wifiPassword[] = "WifiPassword";
const char WebServerInterface::serverAddress[] = "website.com";
const unsigned int WebServerInterface::serverPort = 80;
const char WebServerInterface::serverPostPath[] = "/post.php";
const char WebServerInterface::serverChallengePath[] = "/challenge.php";

// HTTP formatting constants
static const char keyValSeparator = '=';
static const char fieldSeparator = '&';
static const std::string newline = "\r\n";
static const char sessionIdKey[] = "SessionId";

// Authentication MAC constants
static const std::size_t challengeLen = 32;
static const std::uint8_t defaultPaddingByte = 0x00;
```

Compile output for program: DeepCover-Embedded-Security-in-IoT

Figure 3. Web Server Interface source file.

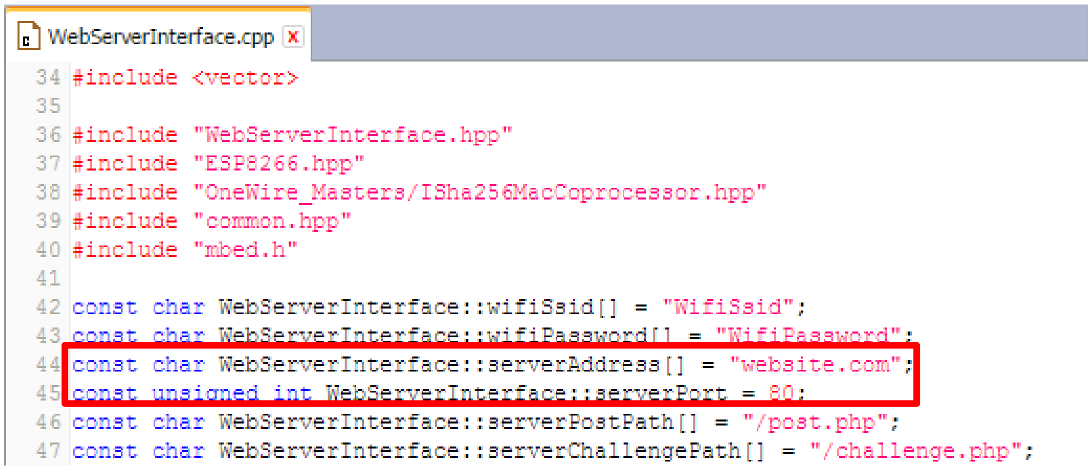
4. Edit the Wi-Fi® SSID and password fields in **WebServerInterface.cpp** with the values for a personal network (**Figure 4**). Note that the ESP8266 Wi-Fi module supports WPA2-PSK and WPA-PSK authenticated networks.


WebServerInterface.cpp

```
#include <vector>

#include "WebServerInterface.hpp"
#include "ESP8266.hpp"
#include "OneWire_Masters/ISha256MacCoprocessor.hpp"
#include "common.hpp"
#include "mbed.h"

const char WebServerInterface::wifiSsid[] = "WifiSsid";
const char WebServerInterface::wifiPassword[] = "WifiPassword";
const char WebServerInterface::serverAddress[] = "website.com";
const unsigned int WebServerInterface::serverPort = 80;
const char WebServerInterface::serverPostPath[] = "/post.php";
const char WebServerInterface::serverChallengePath[] = "/challenge.php";
```

Figure 4. Editing the Wi-Fi SSID and password fields for personal web servers.

5. Populate the fields for the server address and port in **WebServerInterface.cpp** with the web server values. If using the shared instance provided by Maxim, these fields are already correctly populated. Go to this address in a web browser to access the web application user interface (**Figure 5**).

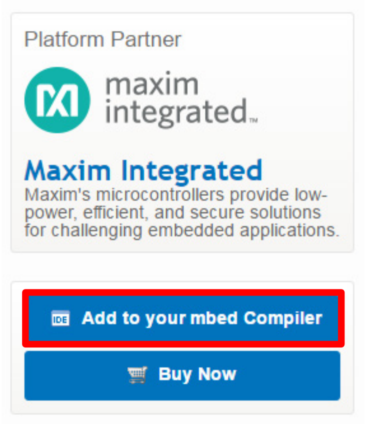

WebServerInterface.cpp

```
#include <vector>

#include "WebServerInterface.hpp"
#include "ESP8266.hpp"
#include "OneWire_Masters/ISha256MacCoprocessor.hpp"
#include "common.hpp"
#include "mbed.h"

const char WebServerInterface::wifiSsid[] = "WifiSsid";
const char WebServerInterface::wifiPassword[] = "WifiPassword";
const char WebServerInterface::serverAddress[] = "website.com";
const unsigned int WebServerInterface::serverPort = 80;
const char WebServerInterface::serverPostPath[] = "/post.php";
const char WebServerInterface::serverChallengePath[] = "/challenge.php";
```

Figure 5. Editing the server address and port fields.

Load Firmware

1. Add the MAX32600MBED platform to your compiler by clicking the **Add to your mbed Compiler** button at the top right side of the [MAX32600MBED platform page](#) (**Figure 6**).

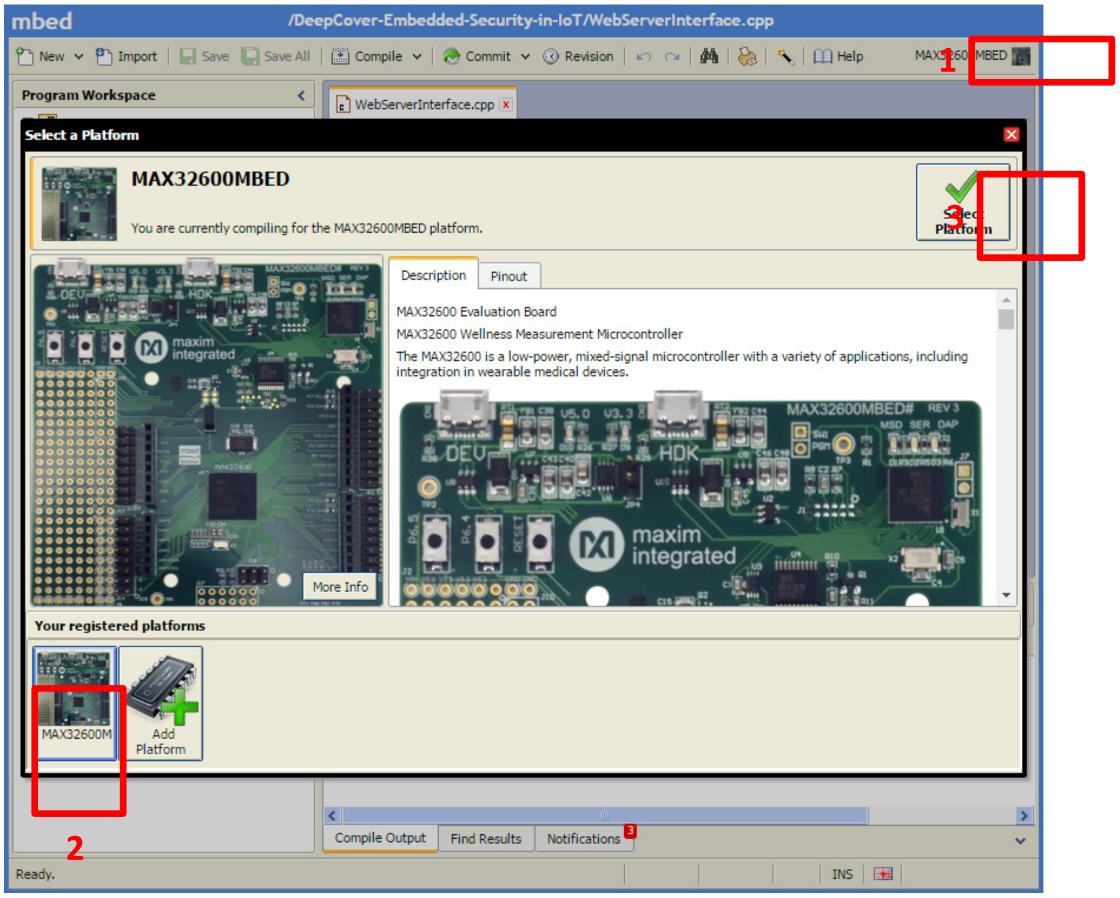

Platform Partner

Maxim Integrated

Maxim's microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications.

Figure 6. "Add to your mbed Complier" button.

2. Return to the online compiler page, and select the MAX32600MBED as the compiler target by using the following steps (**Figure 7**).



You are currently compiling for the MAX32600MBED platform.

MAX32600 Evaluation Board

MAX32600 Wellness Measurement Microcontroller

The MAX32600 is a low-power, mixed-signal microcontroller with a variety of applications, including integration in wearable medical devices.

Figure 7. Selecting the MAX32600MBED as the compiler target.

3. Compile the firmware (**Figure 8**). A window prompts the user to download **DeepCover-Embedded-Security-in-IoT.bin**, the compiled binary firmware.

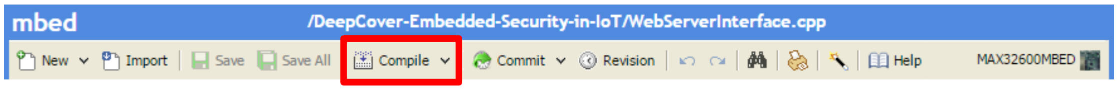


Figure 8. Compiling the firmware.

4. Connect the MAX32600MBED to the computer through the HDK USB port for programming (**Figure 9**).

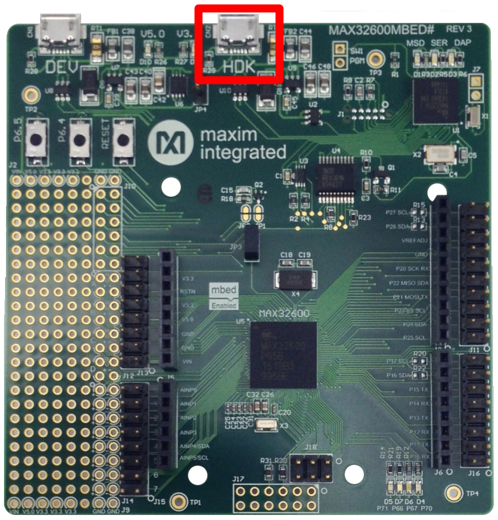


Figure 9. Connecting the MAX32600MBED to the computer through the HDK USB port.

5. The MAX32600MBED acts as a USB storage drive when connected to the computer. Drag the **DeepCover-Embedded-Security-in-IoT.bin** binary to the **MBED** USB storage drive to program the MAX32600MBED.

Run the Demo

During the demo, the mbed shield is referred to as the **Controller**, and the Protected Sensor Node module is referred to as the **Sensor Node**.

Setup

1. Insert the ESP8266 module into its socket as shown (**Figure 10**).

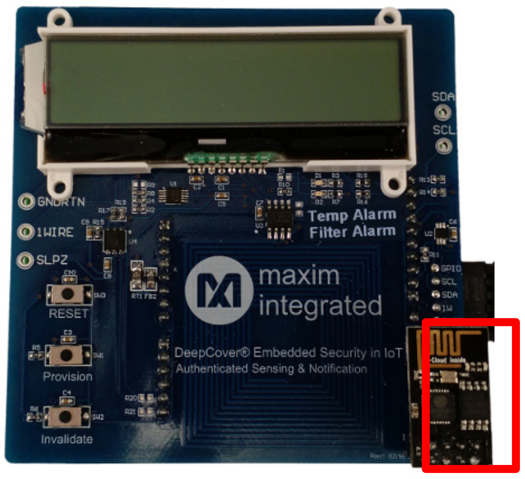


Figure 10. Inserting the ESP8266 module.

2. Insert the downward facing pins on the MAXREFDES143# assembly into the MAX32600MBED# as shown (**Figure 11**).

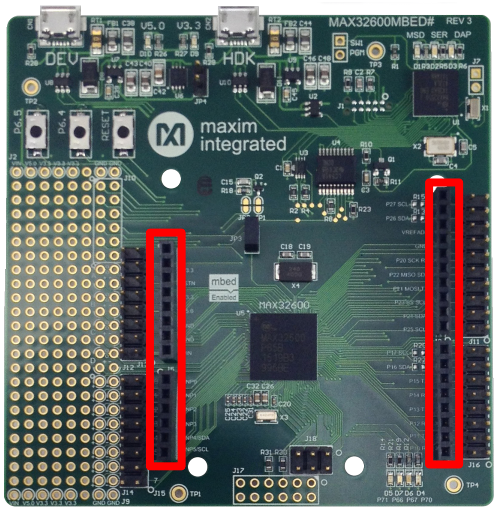


Figure 11. Inserting pins on the MAXREFDES143# assembly into the MAX32600MBED#.

3. Power the MAX32600MBED through the DEV USB port. The ESP8266 immediately begins connecting to the specified Wi-Fi network (**Figure 12**).

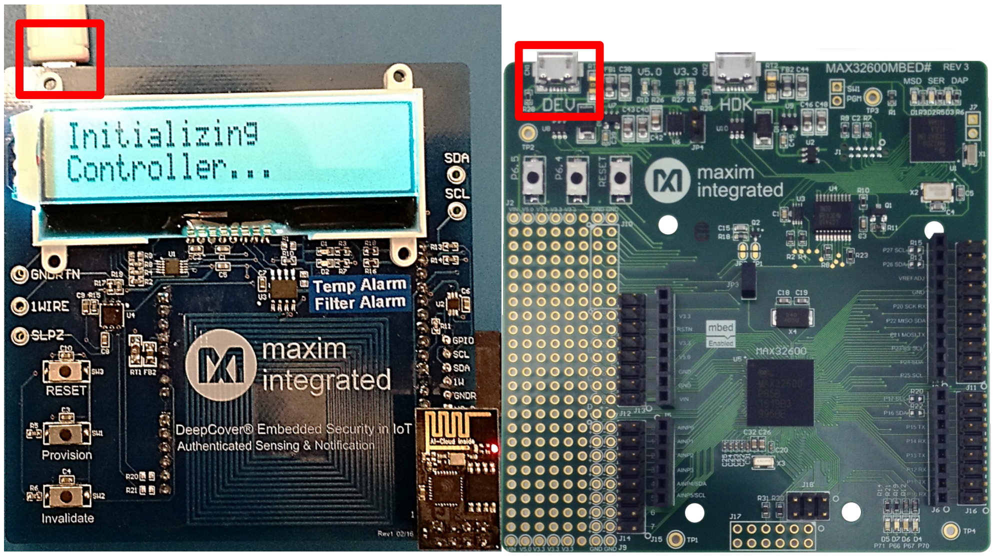


Figure 12. Connecting to the Wi-Fi network.

4. After a successful network connection is established, the display shows the unique ID for the Controller (**Figure 13**). This is used for session tracking on the Web Server. Write down this session ID for later, and then press the **Provision button** to continue.

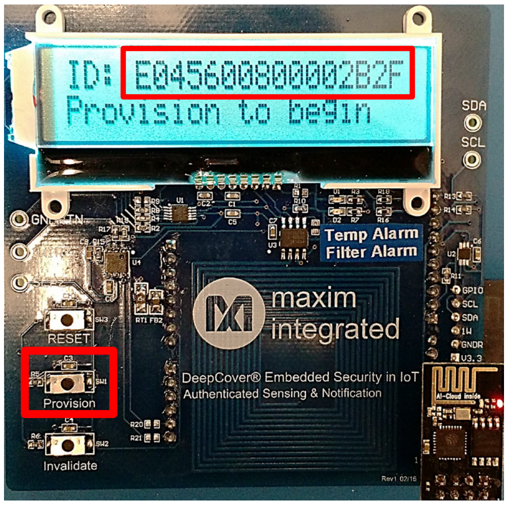


Figure 13. Display for Controller ID.

5. Insert the Sensor Node (1), and then press the **Provision button** (2) to continue (**Figure 14**). A prompt appears if the Sensor Node has never been provisioned. Press the **Provision button** to perform the procedure and continue. The Controller enters **Normal Operation** mode if the Sensor Node was just provisioned or is an existing authentic node. The Controller enters **Sensor Node Not Authentic** mode if the Sensor Node is an existing non-authentic node.

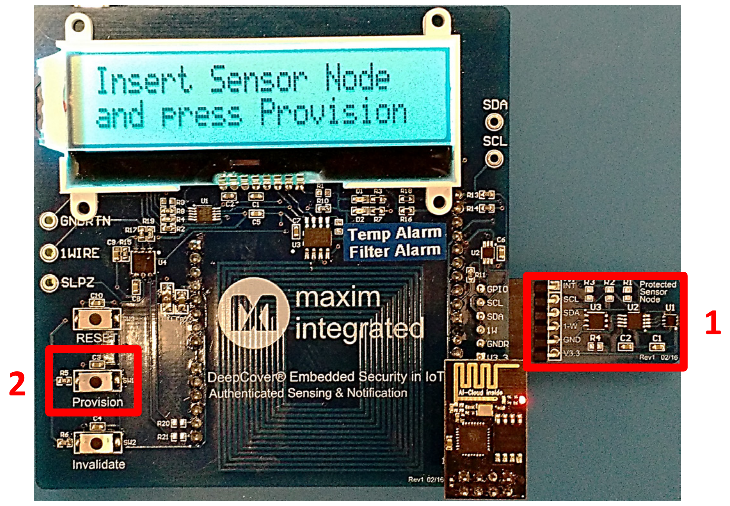


Figure 14. Inserting the Sensor Node.

6. A prompt displays if the Sensor Node has never been provisioned. Press the **Provision button** to perform the procedure and continue (**Figure 15**).

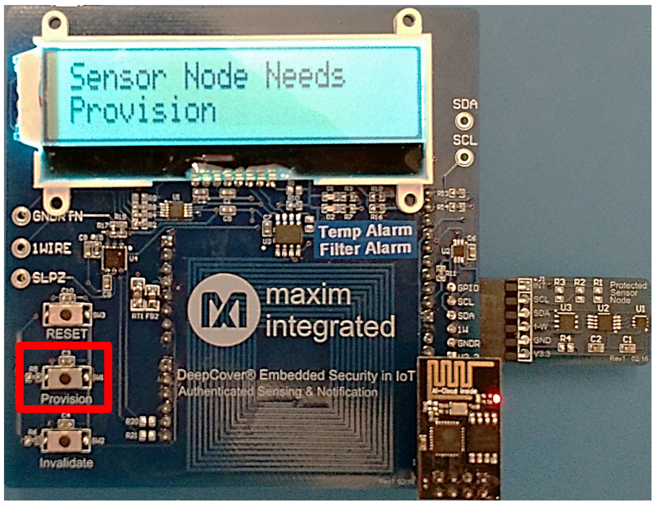


Figure 15. Sensor Node needs Provision button.

Normal Operation

In this mode, the Controller continuously authenticates and takes temperature and filter life measurements from the Sensor Node that represent a consumable water filter within an industrial environment. The latest measurements are shown on the display (**Figure 16**).

The current measurements are sent to the Web Server regularly at approximately 10 second intervals. This interval is configurable by editing file “main.cpp”.

```
static const unsigned int webPostIntervalMs = 10000;
static const unsigned int webPostRetryIntervalMs = 1000;
static const uint8_t maxConsecutiveWebPostErrors = 3;
```

The first filter life measurement is considered to be the full-scale measurement for the remainder of the demo.

The current filter life is the lowest filter life ever observed by the Sensor Node.

A Filter Alarm is registered if the filter life drops to 20%, and a Temperature Alarm is registered if the temperature rises to +26°C. These levels are configurable by editing the SensorData structure in the **common.hpp** file.

```
struct SensorData
{
  static const int8_t tempAlarmLevel = 26;
  static const uint8_t filterLifeAlarmLevel = 20;
```

Press the **Provision button** to return the Sensor Node to repeat the provisioning procedure. This returns the filter life to 100%.

The Sensor Node can also be provisioned with an invalid secret. This causes the Sensor Node to not be recognized as authentic by the Controller, and it will transition to the **Sensor Node Not Authentic** mode. To provision with an invalid secret:

1. Press and hold the **Provision** button.
2. Press and hold the **Invalidate** button.
3. Release the **Provision** button.
4. Release the **Invalidate** button.

Press the **Invalidate button** to enter **Normal Operation with an Invalid Controller**. The LEDs blink once. This mode causes the Controller to use an invalid secret to hash the data sent to the Web Server. This causes the Controller to not be recognized as authentic by the Web Server. Pressing the **Invalidate button** again returns the system back to **Normal Operation** using a valid secret. The LEDs blink twice.

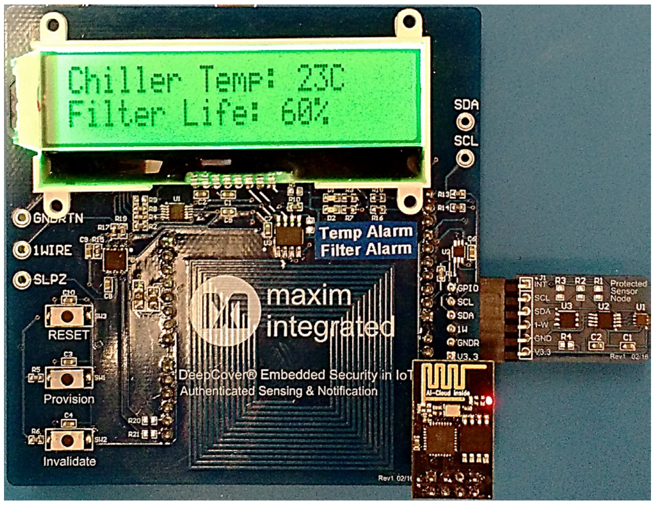


Figure 16. Normal Operation.

Sensor Node Not Authentic

The Controller posts to the Web Server once to indicate that an invalid Sensor Node is detected. After a delay, the demo returns to **Step 5** of the Run the Demo section, **Setup** subsection, and Sensor Node detection repeats.

Pressing the **Provision button** during the delay causes the Sensor Node to be provisioned in the same way as during **Normal Operation**. The demo then returns to **Normal Operation**.

Hardware Error

A Controller Hardware Error indicates that a fault occurred with the shield hardware. This error is not recoverable and the MAX32600MBED must be reset.

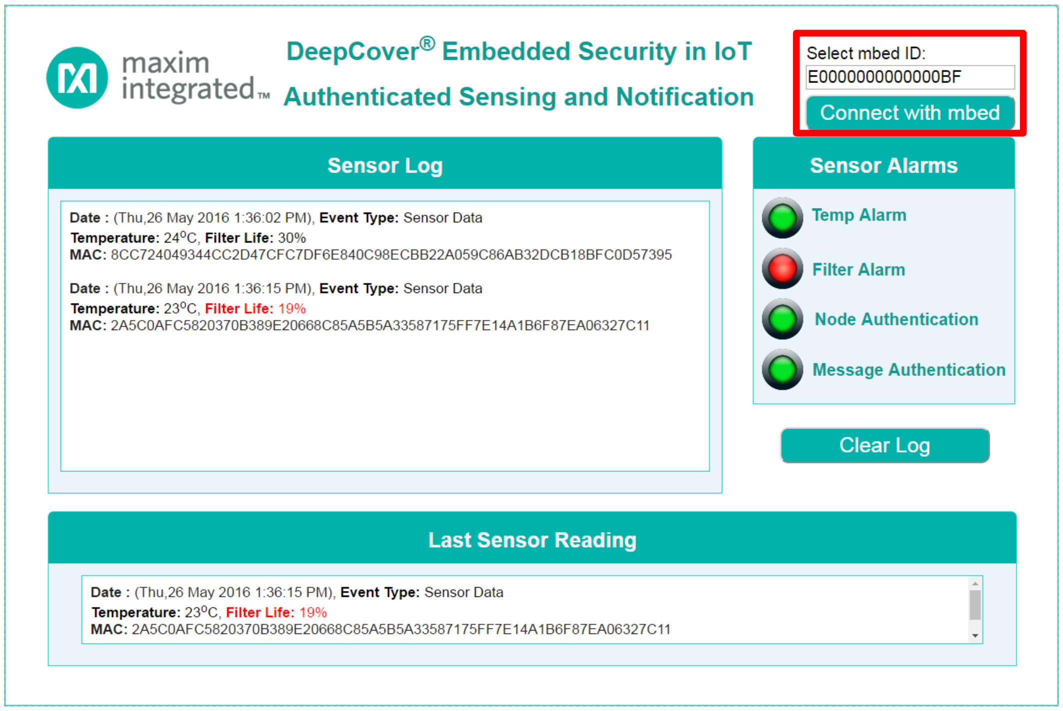
A Sensor Board Hardware Error indicates that a fault occurred with the Protected Sensor Node module hardware. This error is not recoverable and the MAX32600MBED must be reset.

Viewing the Web Server Log

A web-based user interface to the data posted by all shields is included with the demo.

Once a Controller has posted data to the website, it has an individual log created for it on the Web Server. This can be accessed by entering the session ID displayed at the beginning of the demo in the corresponding field of the user interface. (**Figure 17**)

The **Clear Log** button can be used to delete all previous log entries on the web server for the currently selected session ID. This change will clear the “Sensor Log” area of the user interface for anyone viewing that session ID.



Select mbed ID: E0000000000000BF

```
Date : (Thu,26 May 2016 1:36:02 PM), Event Type: Sensor Data
Temperature: 24oC, Filter Life: 30%
MAC: 8CC724049344CC2D47CFC7DF6E840C98ECBB22A059C86AB32DCB18BFC0D57395

Date : (Thu,26 May 2016 1:36:15 PM), Event Type: Sensor Data
Temperature: 23oC, Filter Life: 19%
MAC: 2A5C0AFC5820370B389E20668C85A5B5A33587175FF7E14A1B6F87EA06327C11
```

```
Date : (Thu,26 May 2016 1:36:15 PM), Event Type: Sensor Data
Temperature: 23oC, Filter Life: 19%
MAC: 2A5C0AFC5820370B389E20668C85A5B5A33587175FF7E14A1B6F87EA06327C11
```

Figure 17. Viewing the Web Server log.

Trademarks