

# Arrow IoT Seminar

## LAB MANUAL



## Important Notice

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Copyright © 2014 Texas Instruments Incorporated

## Revision History

May 2014 – Revision 1.0

July 2014 – Revision 2.0

July 2014 – Revision 2.1

August 2014 – Revision 2.2

August 2014 – Revision 2.3

September – Revision 2.4

## Mailing Address

Texas Instruments  
Training Technical Organization  
6500 Chase Oaks Blvd Building 2  
M/S 8437  
Plano, Texas 75023

# CC3200 Tools Setup

---

## Introduction

There are two main sections to this lab:

1. Pre-work to be completed prior to the seminar.
2. Final lab steps to install the CC3200 LaunchPad driver.

The pre-work includes downloading and installing several software packages from the Texas Instruments website. It also includes the steps to import and build some Code Composer Studio projects that are needed for the CC3200 software examples in the Software Development Kit (SDK).

The final steps of the lab require you to plug in a CC3200 LaunchPad board. Since you will receive this board at the seminar, this section of the lab will be performed at the seminar.

## Timing

The pre-work for this lab should take from 45-60 minutes to complete, depending on the speed of your internet connection. The final steps at the workshop should take only 5-10 minutes, once you receive your CC3200 LaunchPad board.

## Prerequisites

To run these labs you will need a Microsoft® Windows® 7 laptop with a good internet connection.

Optional: Android or iOS smartphone to install SimpleLink Wi-Fi Starter application

## Chapter Topics

<b>CC3200 Tools Setup .....</b>	<b>1-1</b>
<i>CC3200 Software Development Kit (SDK) .....</i>	<i>1-3</i>
<i>SimpleLink Wi-Fi Starter Application.....</i>	<i>1-4</i>
<i>Install Arrow IoT Seminar Files .....</i>	<i>1-6</i>
<i>Install Code Composer Studio (CCS) .....</i>	<i>1-7</i>
Import and Build Four Essential CCS Projects .....	1-10
Set the Default Target Configuration .....	1-14
<i>Configure the CC3200 LaunchPad Board .....</i>	<i>1-16</i>
Check CC3200 LaunchPad Jumpers.....	1-16
Install USB Driver .....	1-17
Trouble Installing FTDI Drivers? .....	1-19
Summary .....	1-19

## CC3200 Software Development Kit (SDK)

1. Download the CC3200 SDK version 0.5.2. The seminar labs are built with the version of the SDK. There may be later versions available, but this is the version you will need for the seminar.

► Go to:

[http://software-dl.ti.com/dsps/forms/self\\_cert\\_export.html?prod\\_no=CC3200SDK-0.5.2-windows-installer.exe&ref\\_url=http://software-dl.ti.com/ecs/cc31xx/sdk/cc3200\\_sdk\\_v0\\_5\\_2](http://software-dl.ti.com/dsps/forms/self_cert_export.html?prod_no=CC3200SDK-0.5.2-windows-installer.exe&ref_url=http://software-dl.ti.com/ecs/cc31xx/sdk/cc3200_sdk_v0_5_2)

► You must fill in the U.S. Government export approval fields to download this software. This will be the same procedure for several of the software packages you will need.

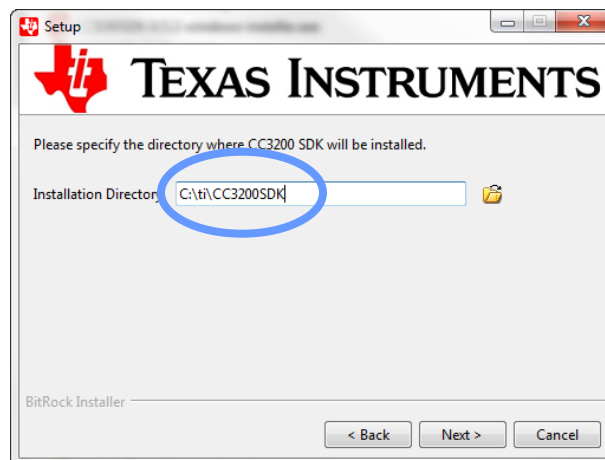
2. Install the CC3200 SDK

► Run the installer and choose the default installation folder `c:\ti\CC3200SDK`

---

**Note:** ► If you already have the CC3200 SDK installed, this installer will over-write your current installation. If you have made any edits/additions to the SDK files, you should re-name your current installation folder.

---

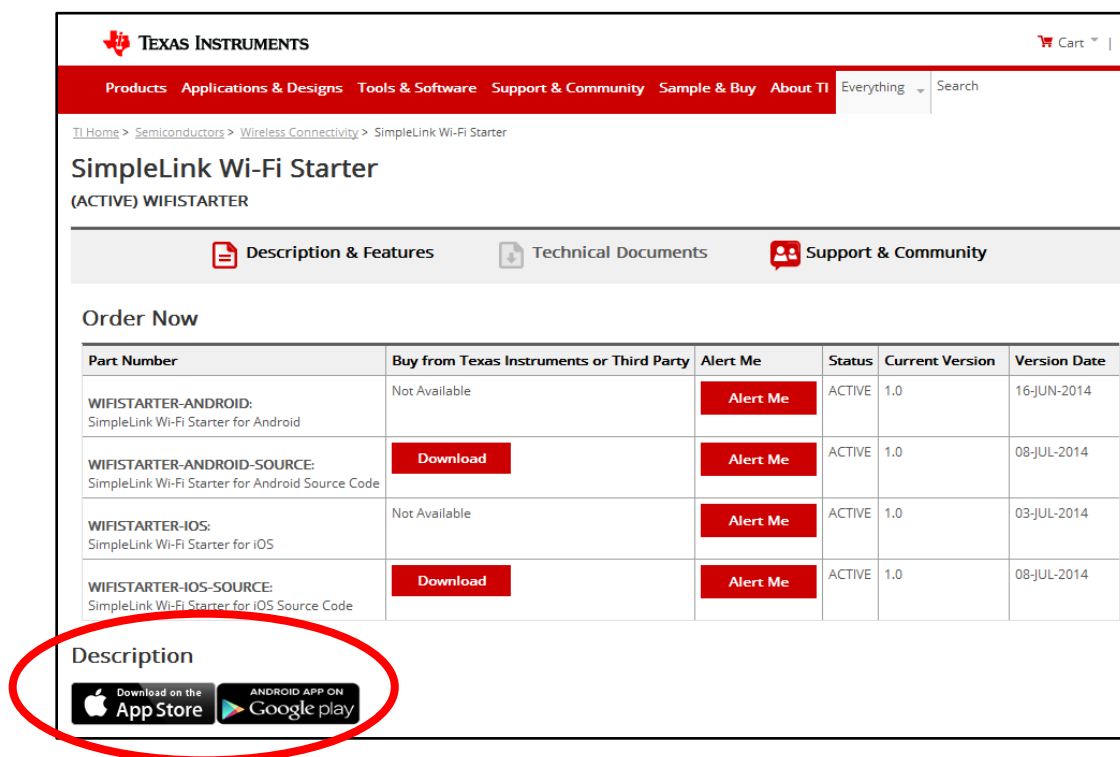


# SimpleLink Wi-Fi Starter Application

If you have an Android or iOS device, download and install the “SimpleLink Wi-Fi Starter Application” from the App Store or Google Play. This application allows you to configure the CC3200s Wi-Fi network information for the first time, using Smart Config (the first tab). A second tab shows you a list of devices that are connected to the Wi-Fi network. A third tab lets you set options to show or hide device information.

## 3. Download and install the application to your Android or iOS Device

- Using your Android or iOS device browser, go to <http://www.ti.com/tool/wifistarter> and click on the appropriate download link (App Store or Google Play)



TEXAS INSTRUMENTS

Products Applications & Designs Tools & Software Support & Community Sample & Buy About TI Everything Search

TI Home > Semiconductors > Wireless Connectivity > SimpleLink Wi-Fi Starter

## SimpleLink Wi-Fi Starter

(ACTIVE) WIFISTARTER

Description & Features Technical Documents Support & Community

### Order Now

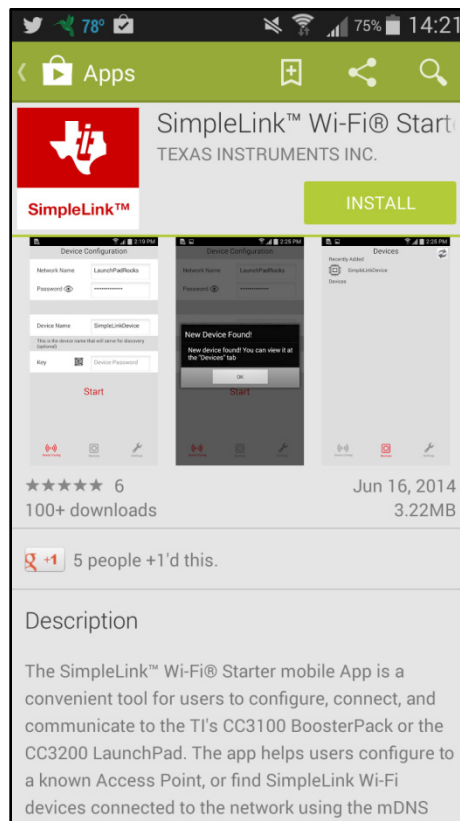
Part Number	Buy from Texas Instruments or Third Party	Alert Me	Status	Current Version	Version Date
WIFISTARTER-ANDROID: SimpleLink Wi-Fi Starter for Android	Not Available	<a href="#">Alert Me</a>	ACTIVE	1.0	16-JUN-2014
WIFISTARTER-ANDROID-SOURCE: SimpleLink Wi-Fi Starter for Android Source Code	<a href="#">Download</a>	<a href="#">Alert Me</a>	ACTIVE	1.0	08-JUL-2014
WIFISTARTER-IOS: SimpleLink Wi-Fi Starter for iOS	Not Available	<a href="#">Alert Me</a>	ACTIVE	1.0	03-JUL-2014
WIFISTARTER-IOS-SOURCE: SimpleLink Wi-Fi Starter for iOS Source Code	<a href="#">Download</a>	<a href="#">Alert Me</a>	ACTIVE	1.0	08-JUL-2014

### Description

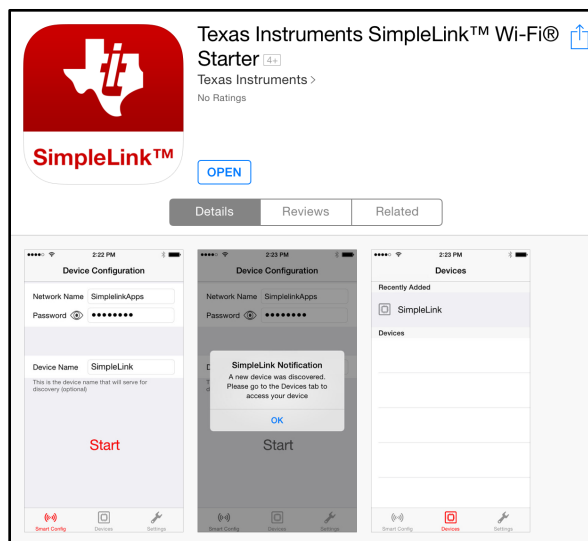
[Download on the App Store](#) [ANDROID APP ON Google play](#)

Note that you can also download the source code for either application from this page but you won't need that for the seminar.

- For the Android app you will see this. Click on the INSTALL button to install.



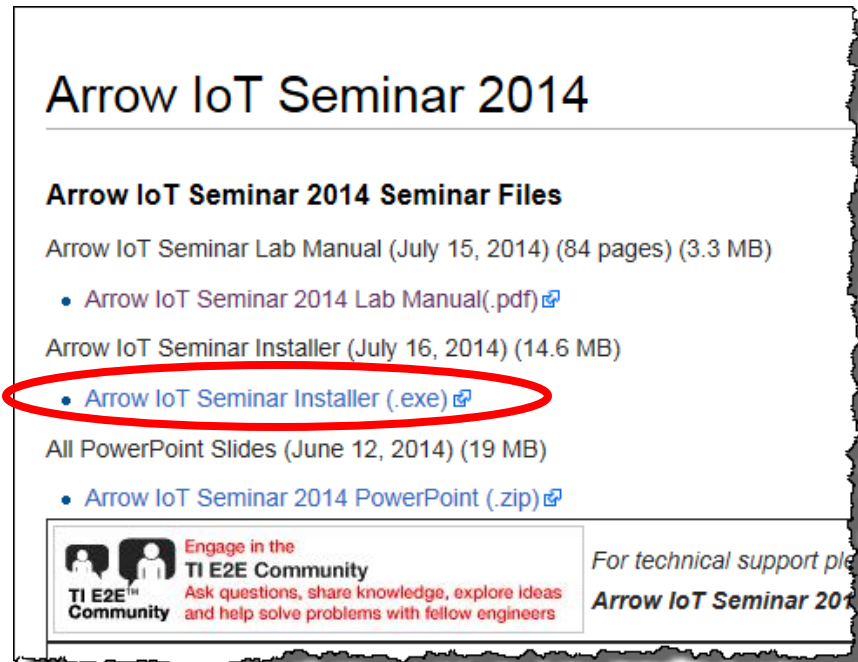
- For the iOS app you will see this. Click on the INSTALL button to install. Note: In this screenshot, the app was already installed, so you see the OPEN button instead.



## Install Arrow IoT Seminar Files

### 4. Download the Arrow IoT Seminar Installer

► Go to [http://processors.wiki.ti.com/index.php/Arrow\\_IoT\\_Seminar\\_2014](http://processors.wiki.ti.com/index.php/Arrow_IoT_Seminar_2014) and click on the Arrow IoT Seminar Installer download link.



### 5. Run the Arrow IoT Seminar Installer

► Run the installer and choose the default installation folder `c:\Arrow_IoT_Seminar`

This will install the project files we'll use in the seminar, along with some documentation files.



# Install Code Composer Studio (CCS)

## 6. Download Code Composer Studio (CCS) version 6.0.1.00040 or later

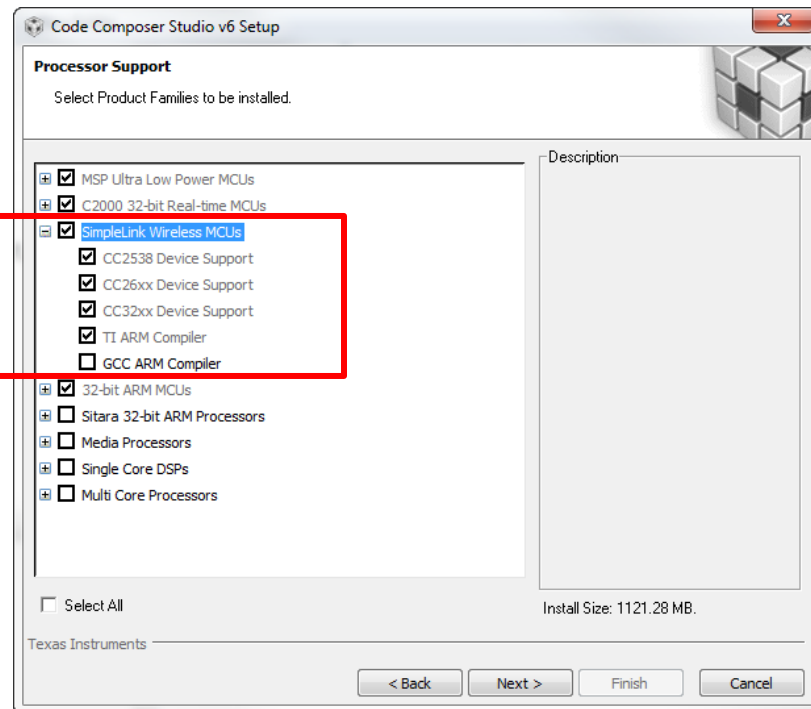
► Go to [http://processors.wiki.ti.com/index.php/Category:Code Composer Studio v6](http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v6) and click on the Windows Download link.

## 7. Install CCS

► Run the installer program and when prompted for Processor Support, select “**SimpleLink Wireless MCUs**”. For the remaining options, select the default values. You should also temporarily turn off your anti-virus software for this installation. The installation time is typically 20 minutes, but can vary based on internet connection speed. If you are using additional processors you can install those as well.

### NOTE:

*This is the only processor support used in the seminar, although you can choose others if you need them. You can also install other processors later if needed.*



**Hint:** If you previously installed CCS version 6.0.1.00040 (or later) and you do NOT yet have SimpleLink Wireless MCUs support installed, you have two choices:

1. We recommend that you re-install CCS. You can just delete or rename the `c:\ti\ccsv6` folder, and then re-install CCS with the processor support you need, including the Wireless Connectivity MCUs. This has the advantage of giving you a clean installation, and might be more reliable than the next option.

2. You can keep your current installation, and just install the Wireless Connectivity MCU support on top of your existing CCS. With CCS v6, this seems to work in most cases. You can't remove existing installed processor support very easily, but it is possible to add new support. Simply run the installer and check the box for Wireless Connectivity MCUs.

**Note:** The link we showed in the previous step is for the “online” installer. If you have any problems installing using this link, you can try downloading a zip file of the full “Off-line” installer from:  
[http://processors.wiki.ti.com/index.php/Download\\_CCS#Code\\_Composer\\_Studio\\_Version\\_6\\_Downloads](http://processors.wiki.ti.com/index.php/Download_CCS#Code_Composer_Studio_Version_6_Downloads).

### Code Composer Studio Version 6 Downloads

There are two types of installers:

- **Web installers** will allow you to perform an install using an installer controlled download process that will only download needed software components. An internet connection is mandatory at install time.
- **Off-line installers** are a large archive (about 730MB). When you run it you can select the components to be installed. No internet connection is required at install time. The executable can be used for installing multiple local systems.

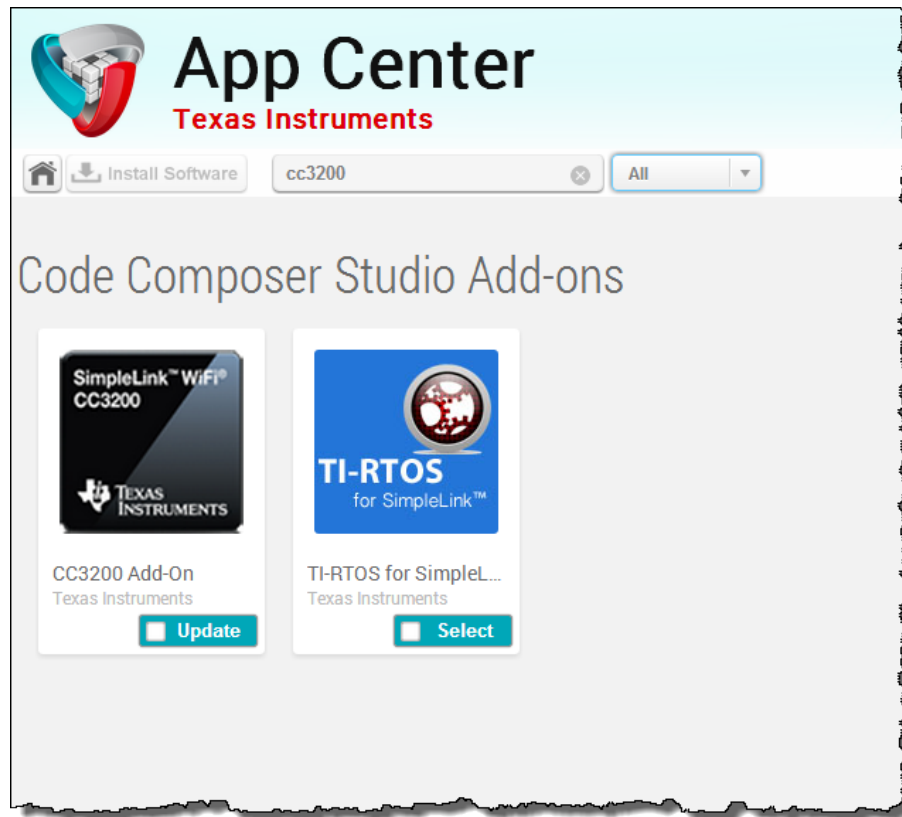
If you have an issue with the web installer not being able to connect to the internet then please try the off-line installer. If you need to update a computer that does not have internet access then download the offline installer and then transfer it to the computer without access and use the offline installer to update your installation.

Release	Build #	Date	Download	Notes
6.0.1	6.0.1.00040	Aug 3, 2014	<b>Web Installers:</b> <a href="#">Windows</a> <a href="#">Linux</a>  <b>Off-line Installers:</b> <a href="#">Windows MD5</a> <a href="#">Linux MD5</a>	<ul style="list-style-type: none"><li>• <b>New in this release:</b><ul style="list-style-type: none"><li>• Fixes an installation issue for laptops using NVidia Optimus-based graphics content changes.</li></ul></li><li>• <b>Installation:</b> see instructions in <a href="#">README</a> file. Additional <a href="#">Linux Installation Instructions</a></li><li>• The <a href="#">software manifest</a> lists the software components included in this product.</li><li>• If you wish to update a previous install then please install the update from within CCS downloading this installation and pointing it to your existing install.</li></ul>
6.0.1	6.0.1.00039	July 22, 2014	<b>Web Installers:</b> <a href="#">Windows</a> <a href="#">Linux</a>  <b>Off-line Installers:</b>	<ul style="list-style-type: none"><li>• <b>New in this release:</b><ul style="list-style-type: none"><li>• Support for Hercules RM57L and TMS570LC (Cortex-R4)</li><li>• Support for SimpleLink CC32xx.</li><li>• New MSP430 EnergyTrace tool for profiling application's energy consumption, n device states and determining execution hotspots (statistical function profile)</li></ul></li></ul>

Should you decide to download this zip file, you should un-zip the file to a folder before running the installer.

**8. Install CC3200 Add-on Update and TI-RTOS for SimpleLink add-on from the CCS App Center:**

- ▶ Start CCS, and choose a Workspace folder (choose the default folder).
- ▶ Open the App Center from the *Help->Getting Started* screen.
- ▶ Search 'cc3200' in the App Center to find 'CC3200 Add-On and 'TI-RTOS for SimpleLink'
- ▶ Select the update box for CC3200 Add-On and select the box for TI-RTOS for SimpleLink
- ▶ Press 'Install Software'



- ▶ Restart CCS when directed after installation

## Import and Build Four Essential CCS Projects

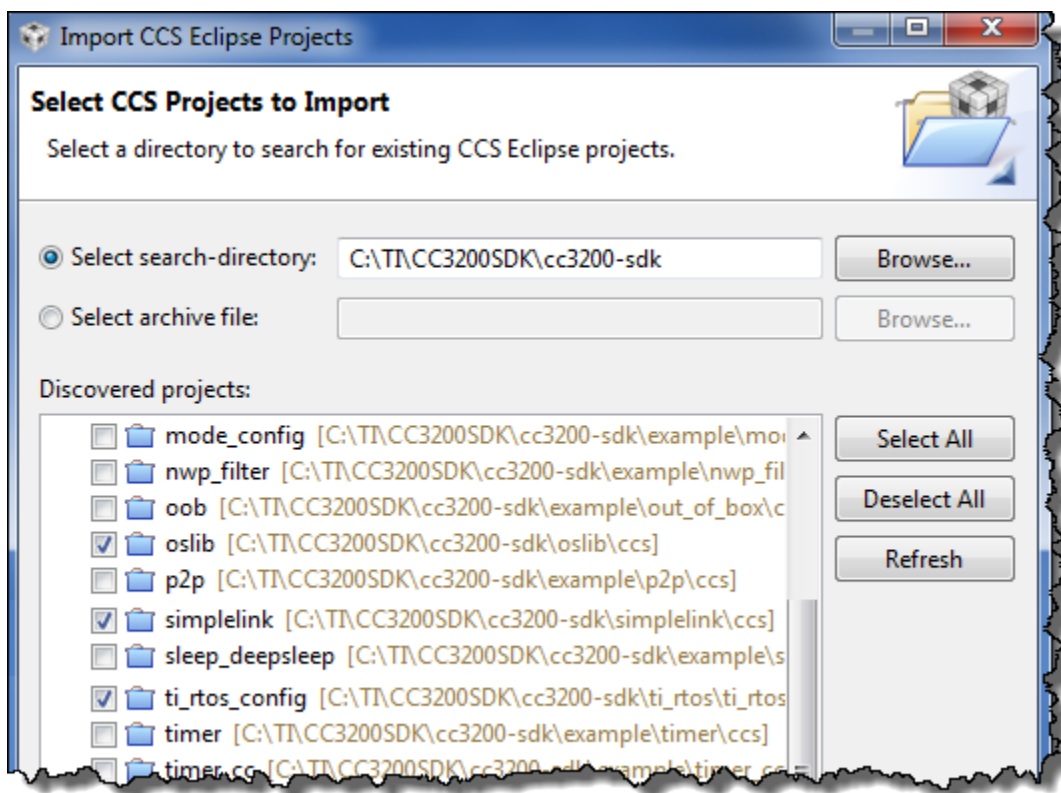
The CC3200 SDK contains four projects that you need to import and build:

- simplelink – creates an object library for SimpleLink APIs
- driverlib – creates an object library for CC3200 peripheral drivers
- oslib – creates the OS adaptation layer library
- ti\_rtos\_config – contains a common TI-RTOS configuration file

It is necessary to build these projects since the libraries and shared TI-RTOS configuration file are used by the CC3200 SDK example projects.

### 9. Import the four projects

- ▶ Choose *Projects>Import CCS Projects* from the Project menu.
- ▶ Select the Browse button in the *Import CCS Eclipse Projects* dialog, and select the directory `c:\ti\CC3200SDK\cc3200-sdk`.
- ▶ Select the four projects by clicking the checkbox for the **driverlib**, **oslib**, **simplelink**, and **ti\_rtos\_config** projects.



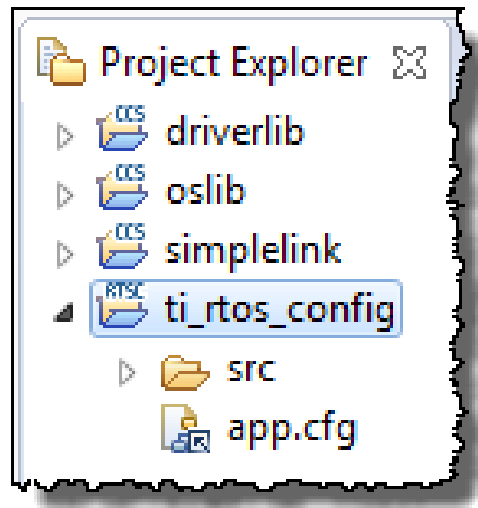
---

**Note:** ▶ Do **NOT** select “Copy projects into workspace” option or “Automatically import referenced projects found in same search-directory”.

---

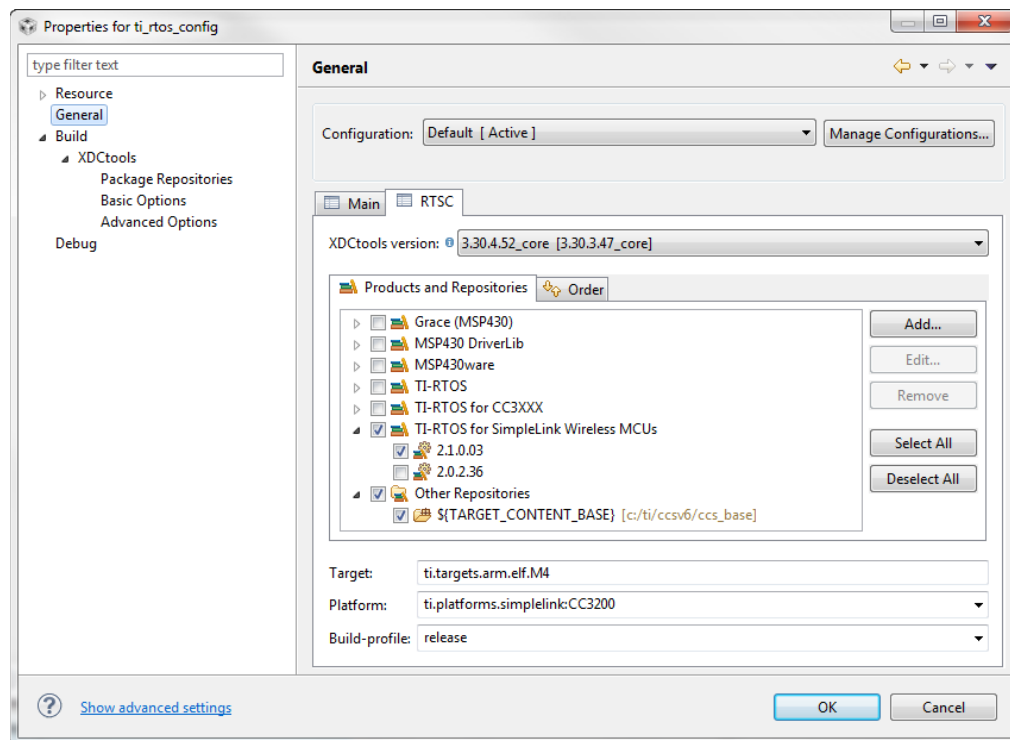
- ▶ After selecting all four projects, click “Finish”.

In the CCS Project Explorer, you should see your four projects listed: **driverlib**, **oslib**, **simplelink**, and **ti\_rtos\_config**.



**10. Set up the *ti\_rtos\_config* project configuration as shown.**

- ▶ Right-click the **ti\_rtos\_config** project name, and select “Properties” from the menu.
- ▶ Click the “General” property in the left column. Click the RTSC tab, and select the latest versions of XDCtools and TI-RTOS for SimpleLink. Also verify the platform is selected as ti.platforms.simplelink:CC3200. Finally, click OK.

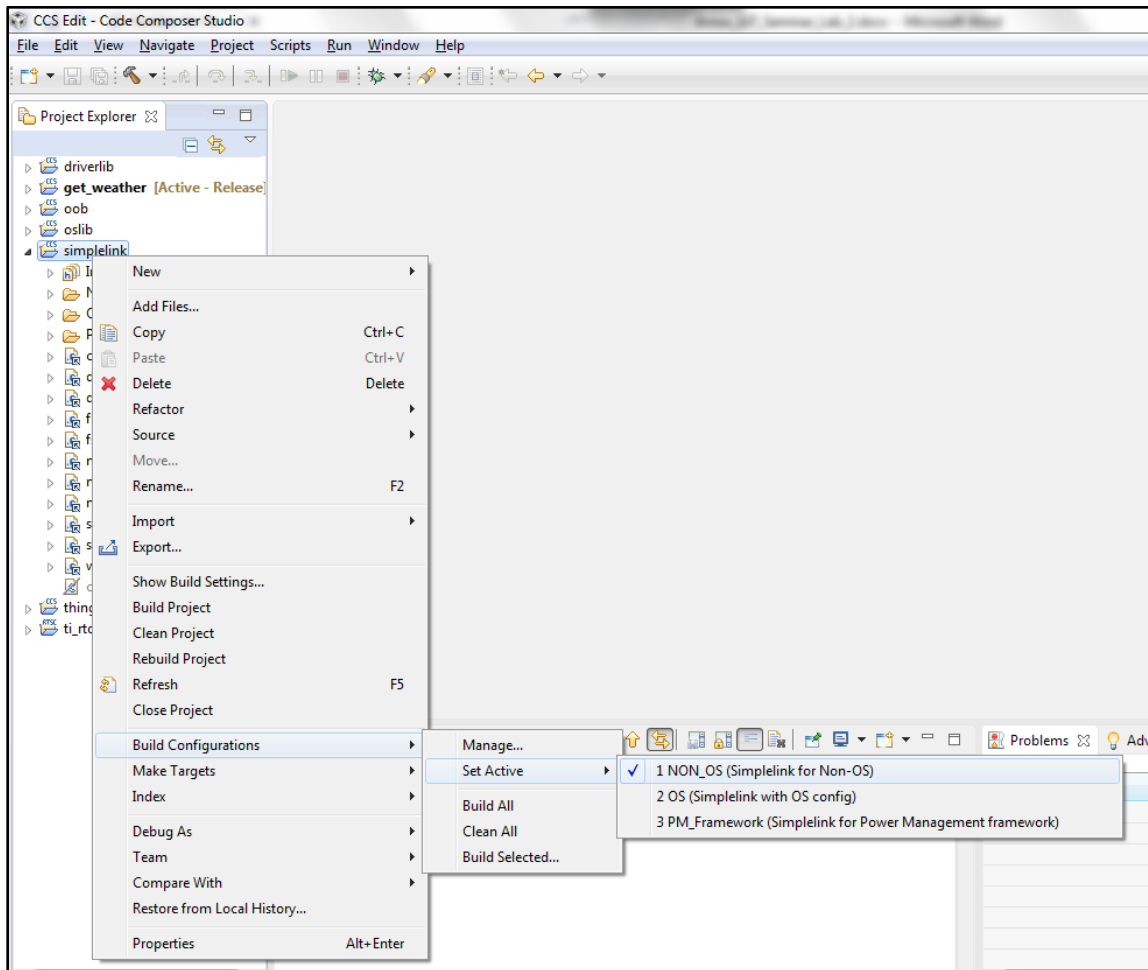


## 11. Build the *simplelink* project for two Build Configurations: NON\_OS and OS

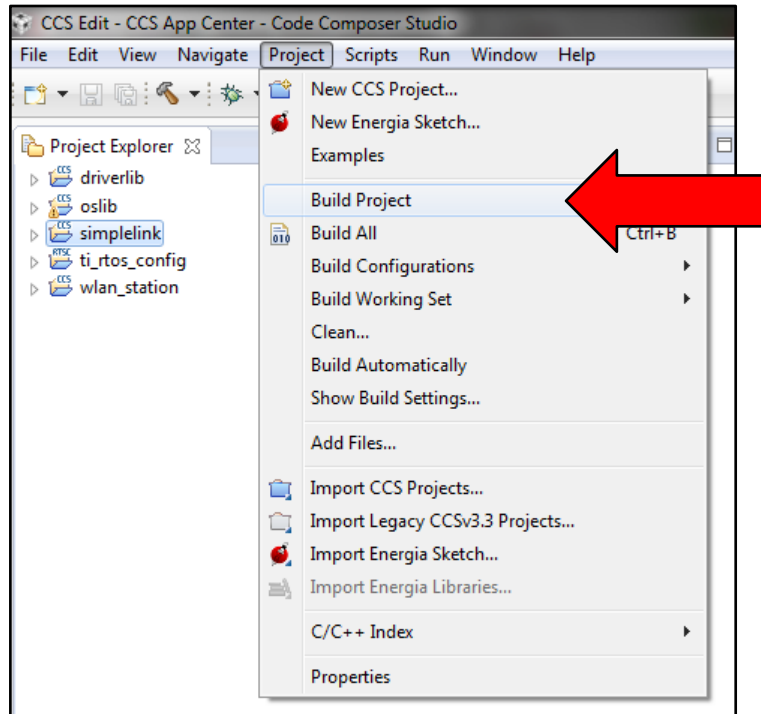
► Right-click the simplelink project, choose Build Configurations, Set Active, and choose:

### 1 NON\_OS (Simplelink for Non-OS).

This selects the NON\_OS configuration to build.



- Select the **simplelink** project and build using the menu command **Project → Build Project**. This builds the NON\_OS configuration.



- Right-click the **simplelink** project again and choose Build Configurations, Set Active, and choose:

## 2 OS (Simplelink with OS config).

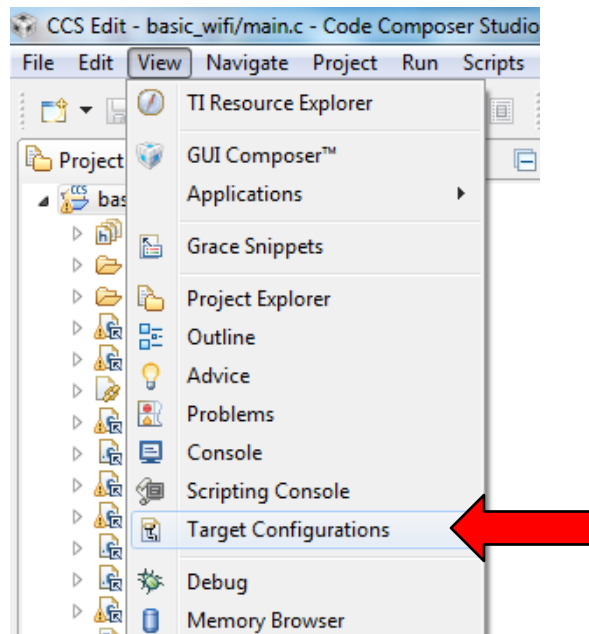
This selects the OS configuration to be active

- Select the **simplelink** project and build this OS configuration
- Select the **ti\_rtos\_config** project and build it.
- Select the **driverlib** project and build it.
- Select the **oslib** project and build it for two Build Configurations: **free\_rtos** and **ti\_rtos**. Note that you may receive two warnings when you build the free\_rtos configuration. You can ignore these warnings.

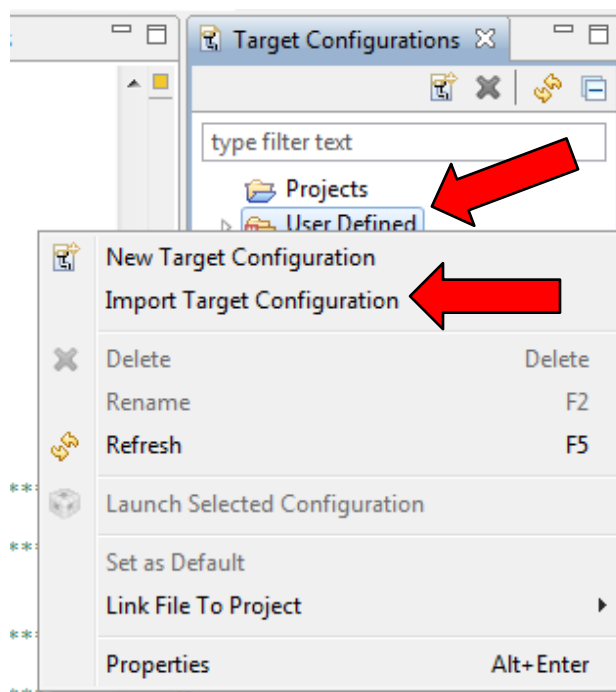
## Set the Default Target Configuration

12. The target configuration needs to be set before debugging from CCS.

- Navigate to **View → Target Configurations**.



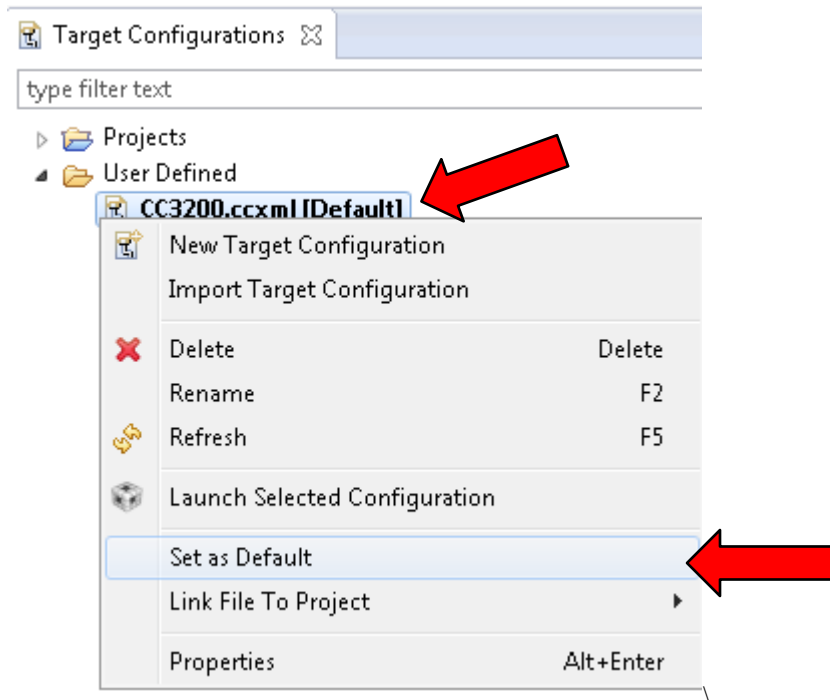
- Right-Click on “User Defined,” select “Import Target Configuration” and navigate to the folder `c:\ti\CC3200SDK\cc3200-sdk\tools\ccs_patch` and select the file `CC3200.ccxml`



- Select the Copy files option when prompted.



- Set this new configuration as the default by right clicking on the file name as shown, and then select “Set as Default”



This is the end of the seminar pre-work. At the seminar you will receive a CC3200 LaunchPad board which is required for the remaining lab steps. If you already have a LaunchPad board, you can proceed with the remaining steps.

## Configure the CC3200 LaunchPad Board

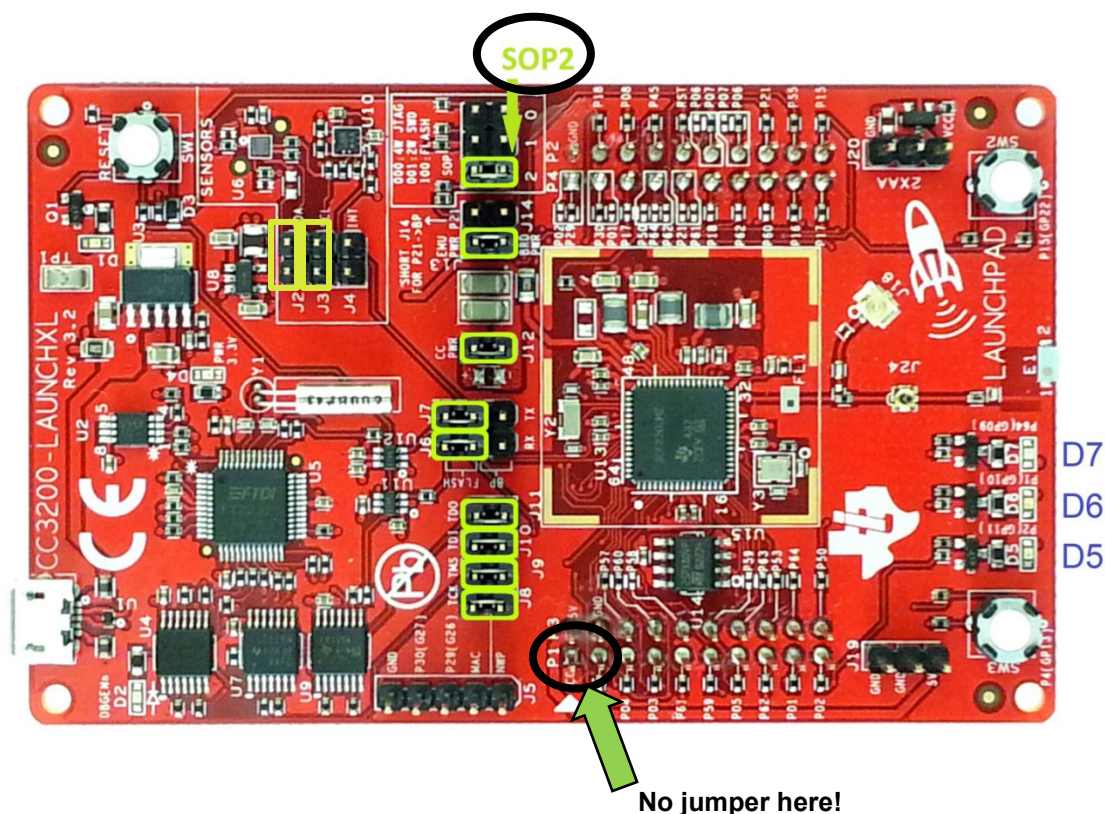
This part of the lab will be done at the seminar you are attending. It requires that you have a CC3200 LaunchPad board. So if you're just doing the pre-work steps, you're done at this point. If you already have your own LaunchPad board, you can go ahead and perform these steps as well.

### Check CC3200 LaunchPad Jumpers

13. The jumpers on the CC3200-LAUNCHXL should be connected as shown.

- Make sure there is NO jumper on P58-VCC
- Make sure the SOP2 jumper has been installed

The jumper on SOP2 (sense on power) is used when debugging CCS projects. This allows downloading code to the CC3200 RAM, and inhibits the CC3200 from automatically boot-loading the application image that is stored in serial flash. This jumper is also used whenever you want to program the serial flash with a new application image, file, digital certificate, or CC3200 service pack. If this jumper is removed, the stored application image is bootloaded from serial flash. On a new board, a pre-programmed "out-of-box" example is loaded and run.



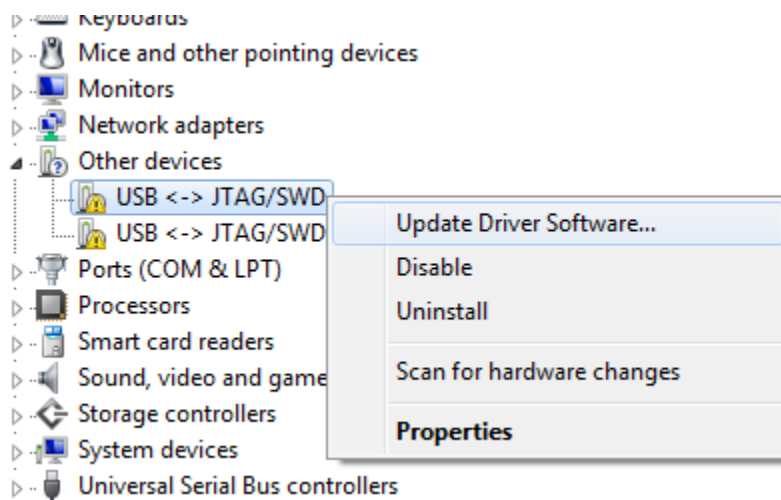
## Install USB Driver

14. The USB drivers may need to be installed manually after connecting your LaunchPad board.

- ▶ Connect the CC3200-LAUNCHXL to the PC using the provided micro-USB cable.
- ▶ Open the Windows Device Manager by selecting **Start Menu→Control Panel→Device Manager**.

If you see two devices in the “Other devices” category named USB <-> JTAG/SWD, you will need to install the driver software for both instances.

If you DON'T see these devices in “Other devices”, check the Ports category to see if the CC3200LP already shows up there. If already installed you can skip the remaining steps.

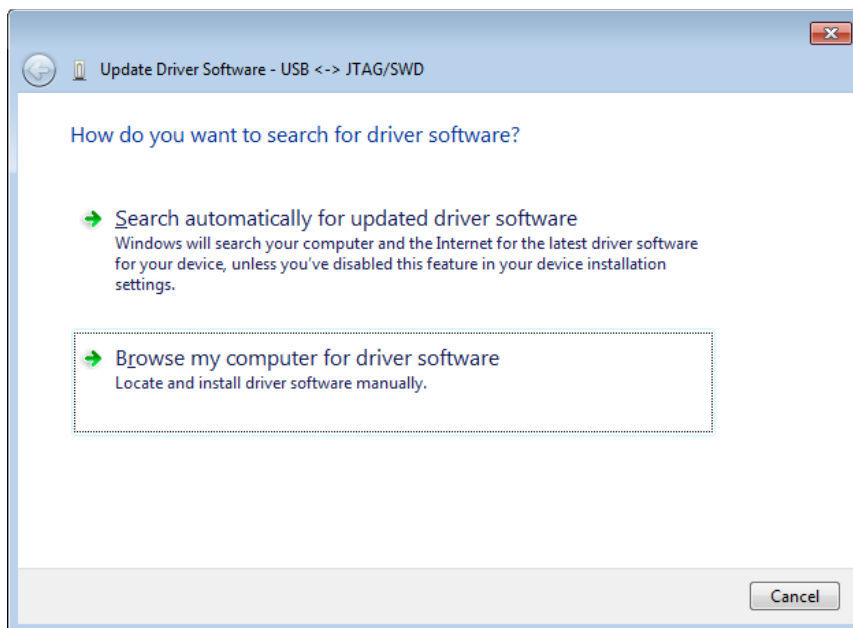


- ▶ Right click on the first instance of “USB <-> JTAG/SWD” and select “Update Driver Software...”
- ▶ Select “Browse my computer for driver software.”

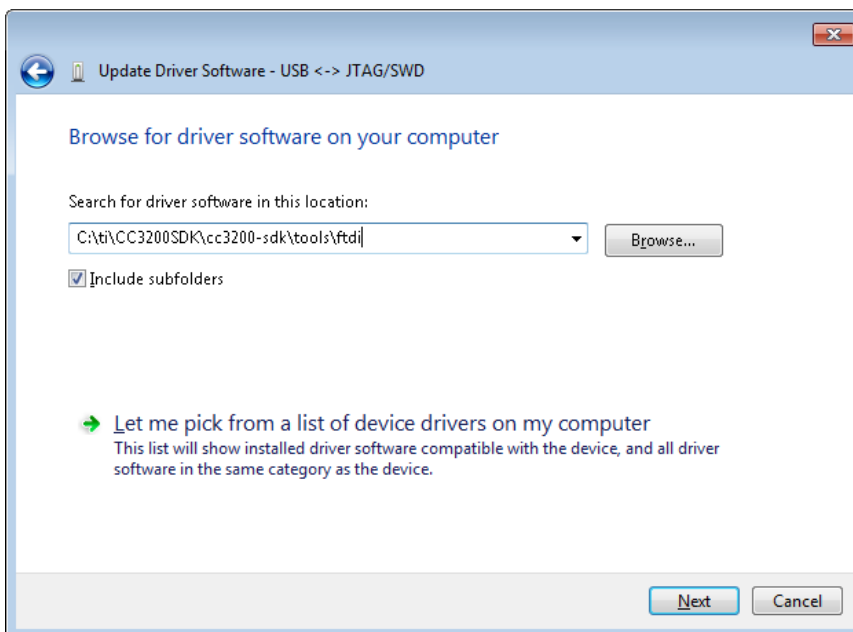
---

**Note:** If you’ve previously installed a TI board that uses FTDI drivers, this process is not necessary.

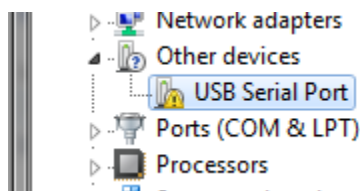
---



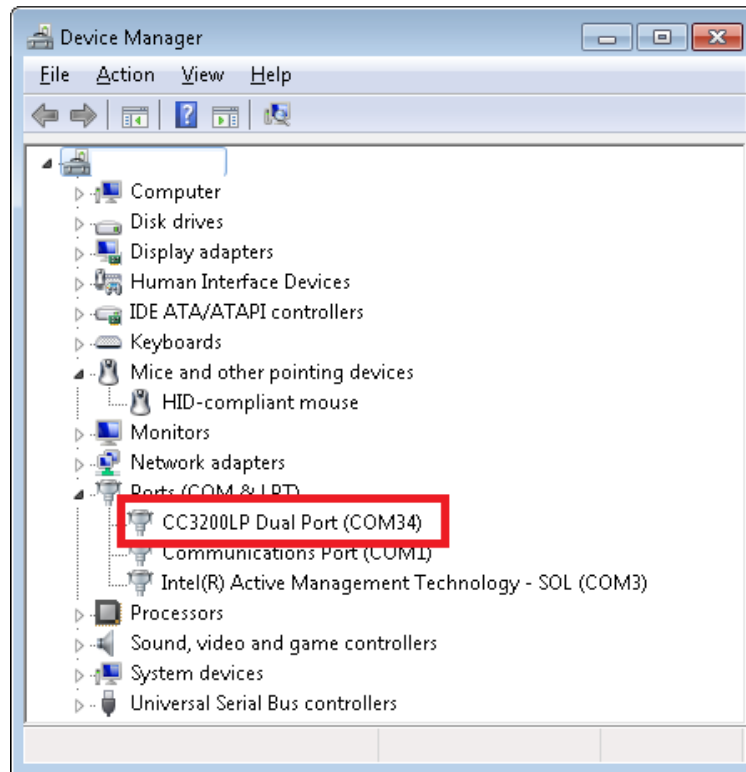
- Browse or fill the search path as `c:\ti\CC3200SDK\cc3200-sdk\tools\ftdi`, and press Next. There is no need to restart the PC.



15. Repeat the above steps for the other instance of "USB <-> JTAG/SWD."
16. Repeat the same steps for the instance of "USB Serial Port" that should have appeared as shown



The CC3200-LAUNCHXL will now be visible in the Device Manager. Note the COM port number that appears.



## Trouble Installing FTDI Drivers?

All new boards should already have the correct FTDI templates flashed into the FTDI chips. However, if your board is not recognized, installing the driver from the SDK is not working, or the built-in FTDI template is corrupted, you'll need to flash the FTDI chip again. Go to this link for instructions to flash the FTDI chip:

[http://processors.wiki.ti.com/index.php/CC31xx %26 CC32xx FTDI Flashing#FTDI Templates](http://processors.wiki.ti.com/index.php/CC31xx_%26_CC32xx_FTDI_Flashing#FTDI_Templates)

## Summary

After the development environment has been set up, see the following resources for further assistance in development:

**CC3200 Programmer's Guide** – This guide contains information on how to use the SimpleLink API for writing WLAN-enabled applications.

**PinMux Tool** – This utility helps determine how to best assign peripherals to the appropriate CC3200 package pins.

**Uniflash** – The Uniflash tool manually stores files on the external serial flash. This includes the application binary and SimpleLink firmware patch files. Also, any configuration files, security certificates, web pages, and so forth can be stored using this tool.

**CC3200 Wiki** – All information and tools for the CC3200, including the above, can be found on the CC3200 Wiki page

(blank page)

# Get Weather Lab

---

## Introduction

This lab assumes you have already installed the CC3200 software tools, and have installed the drivers for the CC3200 LaunchPad.

It also assumes that you have built the four basic projects from the CC3200 SDK – two of which required you to build two separate configurations.

This lab demonstrates how to build and run one of the CC3200 SDK example projects using Code Composer Studio (CCS).

## Lab Objectives

The Get Weather application connects to “openweathermap.org” server, requests the weather details of the specified city, processes the received data, and displays the results in a terminal window.

This application is documented in the SDK file: “c:\ti\CC3200SDK\cc3200-sdk\docs\example\CC32xx Info Center Get Weather Application.pdf” so we won’t repeat the details here.

You will need to edit the `main.c` file in this project to connect to a specific access point, security type, and password, supplied by your instructor. This may vary from location to location, and of course, if using your own access point, you should already know this information.

## Chapter Topics

<b>Get Weather Lab.....</b>	<b>2-1</b>
<i>Lab Steps .....</i>	<i>2-3</i>
<i>Troubleshooting .....</i>	<i>2-9</i>



# Lab Steps

1. **Disconnect your LaunchPad board to power it down.**
2. **Check the jumpers on your LaunchPad board**
  - ▶ Locate a spare jumper in your LaunchPad box
  - ▶ Find the SOP pins on the board, located near the middle-right edge, and insert the SOP2 jumper. For this configuration, the silk screen is labeled 100 FLASH. SOP stands for “sense on power-up”.
  - ▶ Make sure the jumper on P1 (left-most BoosterPack connector) from VCC to P58 is NOT installed. This jumper is used in the out-of-box lab to force the CC3200 to start up in AP (access point) mode. We want this application to come up in STA (station) mode.
3. **Reconnect your LaunchPad board to an available USB port on your computer**
4. **Use Windows Device Manager to determine the COM port for the CC3200 UART. The specific instructions are in the previous lab.**
5. **If not opened already, open Code Composer Studio**
  - ▶ Click the CCS icon on your desktop
  - ▶ Select the default workspace and wait for CCS to start up
6. **Import the Get Weather application**
  - ▶ From the Project menu, select **Project>Import CCS Projects...**
  - ▶ Click the Browse button and navigate to
 

```
c:\ti\CC3200SDK\cc3200-sdk\example\get_weather
```
  - ▶ Do **NOT** check the box Copy Projects into workspace
  - ▶ Click Finish
7. **Edit main.c so the application connects to a specific access point**
  - ▶ Expand the list of folders/files in the get\_weather project by clicking the chevron symbol next to the project name
  - ▶ Double-click the `main.c` file to open it

Near the top of the file, around line 108, find the constants that define the access point name, security type and password. The default values are:


```
#define SSID_NAME      "cc3200demo"      /* AP SSID */
#define SECURITY_TYPE  SL_SEC_TYPE_OPEN  /* Security type (OPEN or WEP or WPA) */
#define SECURITY_KEY   ""                /* Password of the secured AP */
```

- ▶ Change these values to the correct ones. Change the `SSID_NAME` to your access point name. For WPA/WPA2 security, use the constant `SL_SEC_TYPE_WPA`. For `SECURITY_KEY`, enter the password for your access point. At the seminar, your instructor will tell you which SSID and password to use.

```
#define SSID_NAME      "yourSSID"        /* AP SSID */
#define SECURITY_TYPE  SL_SEC_TYPE_WPA   /* Security type (OPEN or WEP or WPA) */
#define SECURITY_KEY   "yourPassword"    /* Password of the secured AP */
```

## 8. Build the Get Weather application

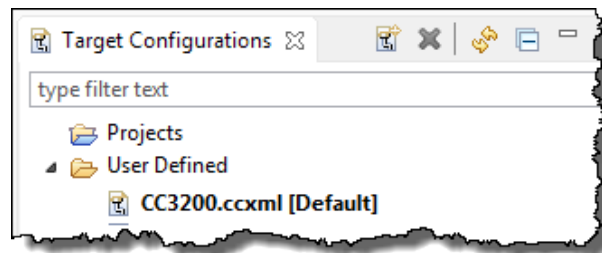
► Select the get\_weather project in the Project Explorer pane. Make sure that it shows that it is the [Active] project (the name will be shown in bold type too)

► Build your project using the “hammer”  icon on the tool bar. When you see \*\*\*\* Build Finished \*\*\*\* in the console, you’re ready to run the application.


► If you get build errors, see some of the troubleshooting tips at the end of this lab.

## 9. Verify you have set the default target configuration (should be already done in previous lab)

Select **View → Target Configurations**, and make sure CC3200.ccxml is your default target configuration.

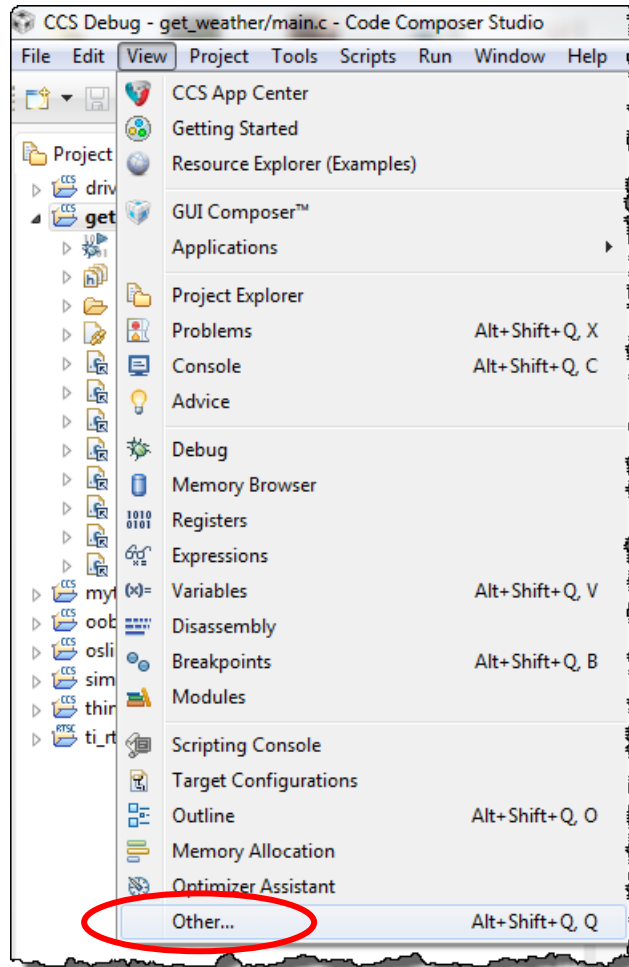


## 10. Debug the Get Weather application

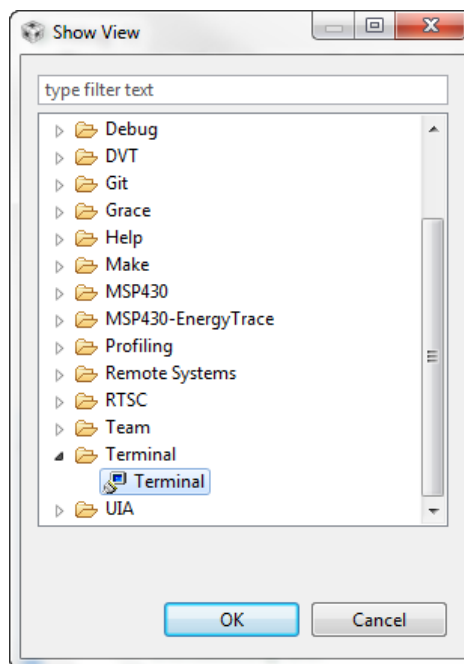
- Click the “bug”  icon on the toolbar to start the debugger


This should start the CCS Debugger, your code should be loaded into the CC3200 RAM, and it should run the initialization code, stopping at the first line of `main()`

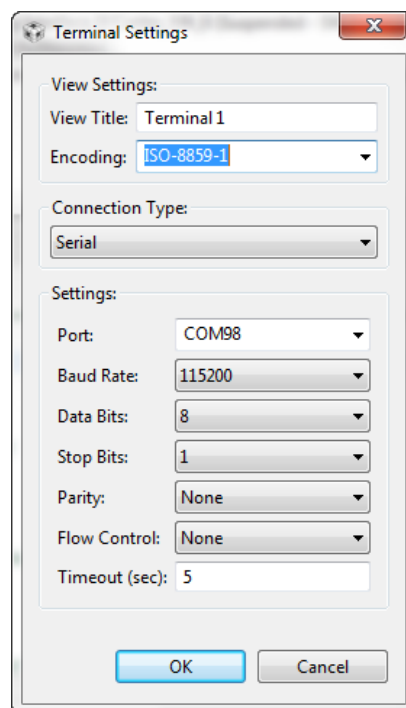
- Open the terminal utility in CCS debug perspective. In the menu, select **View → Other...**





- Scroll down, expand the Terminal folder, and select Terminal

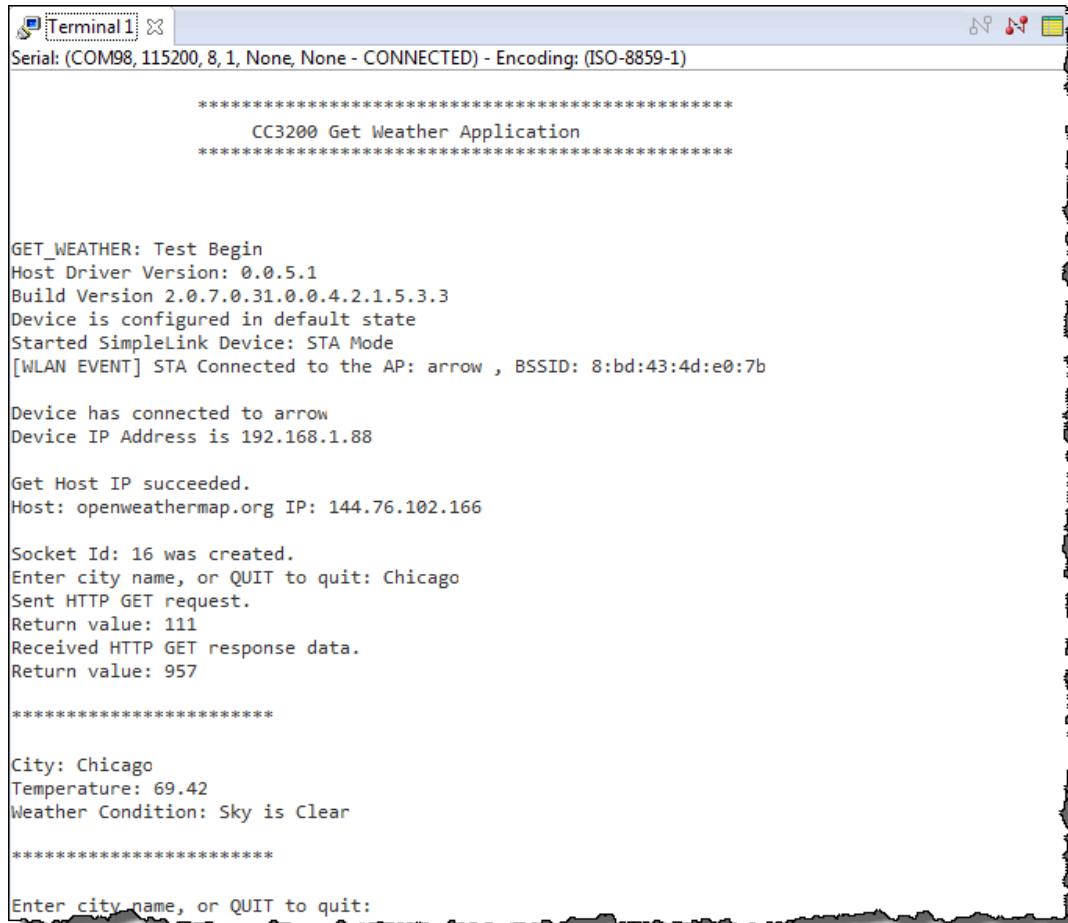


- In the Terminal menu bar, click on the  settings button.
- Select Connection Type: Serial, select your COM port number from the drop-down list, choose other settings as shown:



- When you click OK the terminal should connect to your board. If not, click the connect  button.

- ▶ To run your code, click the Resume  button on the toolbar (this looks like a “Play” button), alternatively you can use the Run menu and select **RUN → Resume**
- ▶ You should see the status of your session in the terminal window.
- ▶ Enter the desired city name, and wait for the response. As we showed earlier in the lab, this is an example of what your terminal screen should look like:



```
Terminal1
Serial: (COM98, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)

*****
CC3200 Get Weather Application
*****

GET_WEATHER: Test Begin
Host Driver Version: 0.0.5.1
Build Version 2.0.7.0.31.0.0.4.2.1.5.3.3
Device is configured in default state
Started SimpleLink Device: STA Mode
[WLAN EVENT] STA Connected to the AP: arrow , BSSID: 8:bd:43:4d:e0:7b

Device has connected to arrow
Device IP Address is 192.168.1.88

Get Host IP succeeded.
Host: openweathermap.org IP: 144.76.102.166

Socket Id: 16 was created.
Enter city name, or QUIT to quit: Chicago
Sent HTTP GET request.
Return value: 111
Received HTTP GET response data.
Return value: 957

*****

City: Chicago
Temperature: 69.42
Weather Condition: Sky is Clear

*****

Enter city name, or QUIT to quit:
```

**Note:** If instead, you are prompted to enter an SSID name, that indicates that the CC3200 could not connect to the access point you specified in the #define constants. You should check your code, and check with your instructor to make sure you have the correct access point name, security type, and password.

- ▶ When you're finished with this lab, enter “quit” in the terminal window

### 11. Terminate the Debug session

On the toolbar click the “big red square” button  labeled “Terminate”, or use (Ctrl+F2). This properly terminates the debug session, disconnects you from the target CC3200 LaunchPad board, and returns you to CCS edit mode

### 12. Close CCS.



That's it. You're done with this lab.

# Troubleshooting

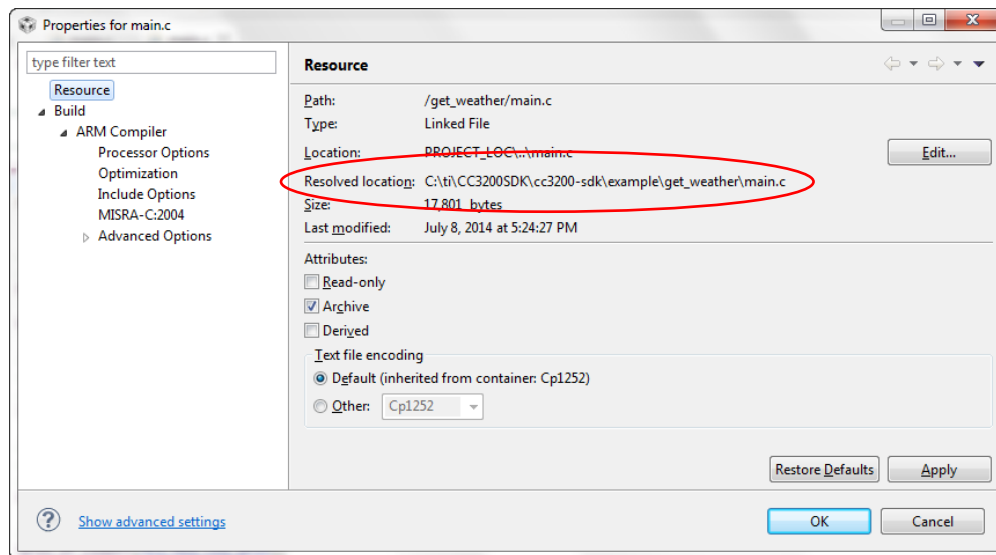
## 1. CCS terminal cannot connect to the COM port

Are you connecting to the CC3200 UART COM port? Check Windows Device Manager  
Check your terminal settings – do you have Flow Control=None?

## 2. I get “build” errors in CCS

Check the error messages in the CCS Problems window

Check the location of the project source files, e.g. main.c. Right-click on the main.c file and verify that the Resolved location is in the SDK folder. If you selected “Copy files into workspace” CCS will not find the source files.



Make sure you have imported and built the SDK projects listed in Lab 1. These include: **simplelink**, **driverlib**, **oslib**, and **ti\_rtos\_config**. See the Lab 1 instructions. A quick way to check this is to expand each project, and check for the following:

**driverlib** project – open the “Release” folder, you should see the “driverlib.a” library file

**oslib** project – open the “free\_rtos” folder, you should see the “free\_rtos.a” library file, also check the “ti\_rtos” folder, you should see the “ti\_rtos.a” library file

**smartlink** project – check the “NON\_OS” folder, you should see the “simplelink.a” library file, also check the the “OS” folder, you should see the “simplelink.a” library file

## 3. I get linker errors when building the project like

```
>> ERROR: mklb: gmake error during rtsw7M4_T_le_eabi.lib build
warning #10207-D: automatic RTS selection: resolving index library
"libc.a" to
"c:\ti\ccsv6\tools\compiler\arm_5.1.7\lib\rtsw7M4_T_le_eabi.lib", but
"c:\ti\ccsv6\tools\compiler\arm_5.1.7\lib\rtsw7M4_T_le_eabi.lib" was
not found
warning #10062-D: entry-point symbol "_c_int00" undefined
```

The problem is that automatic library build step has failed, which causes the C runtime library to not be generated. It is usually caused if your system PATH environment variable has some versions of MKS Toolkit or Cygwin tools installed, or incompatible versions of sh.exe in the PATH, one E2E forum post says it could also be WinAVR. The work-around is to remove

these utilities from your PATH before running CCS. You can do this by going to Control Panel – System – Advanced System Settings – Environment Variables... - System Variables, scroll down to path and click Edit.

**Hint:** A simpler solution would be to simply copy the run-time support library file to the ARM compiler lib folder. We provided the file in the seminar installer and it should also be available on the USB flash drive provided by your instructor.

► Copy the file `c:\Arrow_IoT_Seminar\rtsv7M4_T_le_eabi.lib` to the folder `c:\ti\ccsv6\tools\compiler\arm_5.1.7\lib`. Then re-compile your project.

#### 4. The debugger cannot load my code

Did you set the default target configuration file in Lab 1? If not, Click on *View-Target Configurations*. Then click on “User Defined,” select “Import Target Configuration” and select the file `CC3200.ccxml` from `c:\ti\CC3200SDK\cc3200-sdk\tools\ccs_patch`

Terminate your Debug session (red square box icon). Disconnect LaunchPad, re-connect LaunchPad, and try the Debug icon once again.

#### 5. Application cannot connect to access point

Double check your SSID, security type, and password #DEFINEs in `main.c`

Make sure the access point is available (use your smartphone or PC to see if SSID is broadcasting)

Verify the access point has not reached the maximum number of connected devices, some portable mifi devices can only support 5, 10, or 15 connections. Check with your instructor.

Perhaps there is too much 2.4 GHz traffic/interference in the room. Try setting the access point to use a specific Wi-Fi channel, usually channel 6 or channel 11 is less crowded than channel 1. You could use a Wi-Fi analyzer application on your smartphone to observe the Wi-Fi channel traffic in the room. Check with your instructor on this solution.

#### 6. Application cannot connect to [openweathermap.org](http://openweathermap.org), or DNS error obtaining IP address for external site after connection to access point

If your access point has three DNS servers listed, remove one of the servers, and all should work o.k. – this is a known problem with one of the SimpleLink APIs to be fixed in an upcoming release. It can be found on the E2E forums.

Make sure your access point has internet access.

#### 7. When I enter the city name, it does not respond with a temperature

Not every city is on the site, try a major city. Reload your code and click Resume. In other words, try again.



# ThingFabric Lab

---

## Introduction

This lab demonstrates how to collect sensor data from the sensors on the CC3200 LaunchPad board. This data is then published to the 2lemetry ThingFabric platform using mqtt protocol.

In the Get Weather lab, you needed to modify the #DEFINE constants for the Wi-Fi access point (SSID, security type, and password). In this lab, we'll use SmartConfig technology to create this connection using an Android or iOS application called SimpleLink Wi-Fi Starter.

## Lab Objectives

- Use the SimpleLink Wi-Fi Starter application on your smartphone/tablet to connect your CC3200 LaunchPad board to a Wi-Fi access point. Once connected, the application stores the access point profile in the serial flash, so you will be able to connect automatically in subsequent labs, i.e. this is a one-time setup for storing the profile.
- Connect to 2lemetry ThingFabric platform using a TI demo account created specifically for this seminar.
- Collect sensor data (accelerometer and temperature) on the CC3200 LaunchPad board, and publish this data to the 2lemetry platform using mqtt protocol. The code contains a simple mqtt client.
- Use the CCS terminal window to observe the sensor values being published to the cloud server
- Examine the sensor data at the 2lemetry site using a web portal built for this seminar.

## Chapter Topics

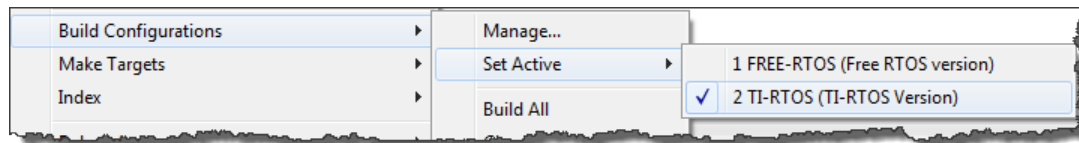
<b>ThingFabric Lab .....</b>	<b>3-1</b>
<i>Import and Build ThingFabric CCS Project.....</i>	<i>3-3</i>
<i>Modify the ThingFabric Application.....</i>	<i>3-5</i>
<i>Connect CC3200 to Access Point using SmartConfig.....</i>	<i>3-8</i>
Observe the terminal messages .....	3-9
<i>Troubleshooting .....</i>	<i>3-11</i>

## Import and Build ThingFabric CCS Project

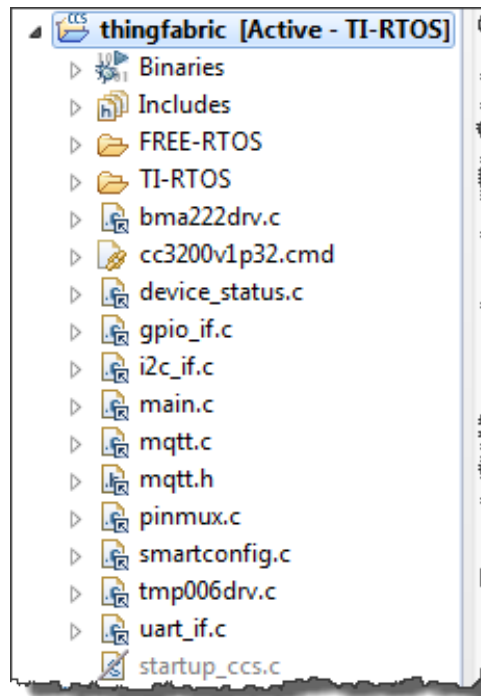
1. As in the previous lab, verify that the SOP2 jumper is installed on your LaunchPad board. Also verify there is no jumper on VCC-P58. We don't want the CC3200 to come up in AP mode.
2. Connect your LaunchPad board to an available USB port on your computer (it may already be connected).
3. Open Code Composer Studio, if not already running.
  - ▶ Click the CCS icon on your desktop
  - ▶ Select the default workspace and wait for CCS to start up
4. Import the ThingFabric application
  - ▶ From the Project menu, select **Project** → **Import CCS Projects...**
  - ▶ Click the Browse button and navigate to `c:\Arrow_IoT_Seminar\thingfabric`
  - ▶ Do **NOT** check the box "Copy Projects into workspace"
  - ▶ Click Finish

## 5. Select TI-RTOS as the Active Build Configuration

- ▶ Right-click on the **thingfabric** project in Project Explorer
- ▶ Click **Build Configurations** → **Set Active**, and select **2 TI-RTOS (TI-RTOS Version)**



- ▶ Expand the list of files in the project. You should see:



## 6. Build the ThingFabric application

We're going to build the existing project, just to ensure that it has been imported properly and that there are no errors.

- ▶ Select the **thingfabric** project in Project Explorer pane. Make sure that it shows that it is the [Active] project (the name will be shown in bold type too)

- ▶ Build your project using the “hammer”  icon on the tool bar.

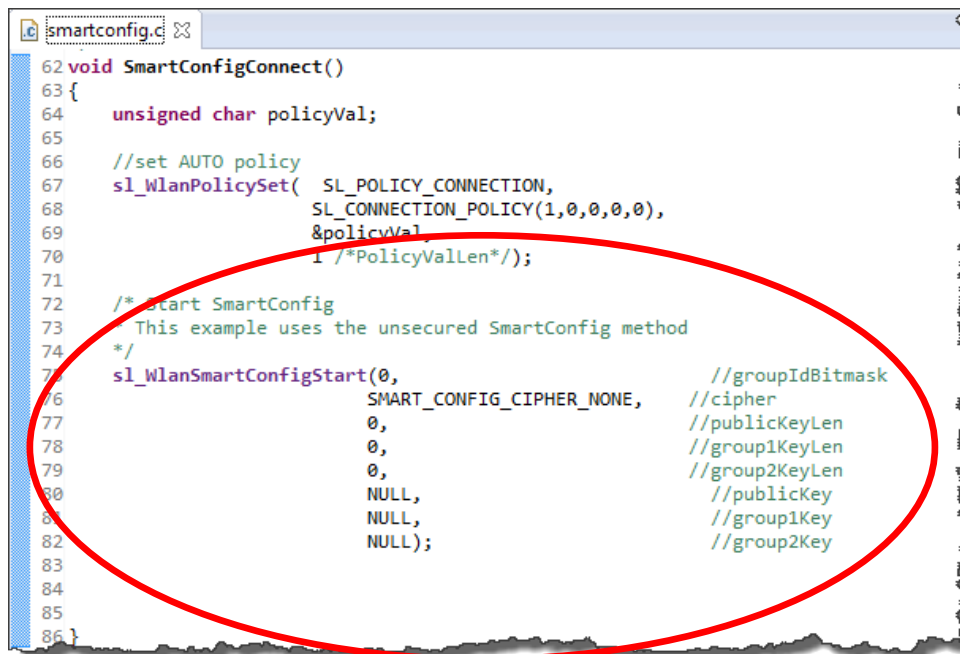
- ▶ If you encounter any build errors, verify that you imported the project correctly. Ask your instructor for help if you can't resolve an error message.

# Modify the ThingFabric Application

## 7. Modify the ThingFabric application to encrypt the password when calling `sl_WlanSmartConfigStart`

**Note:** In this application we want to use TI's SmartConfig technology to connect your CC3200 to an access point. SmartConfig allows multiple devices to connect using a single click. In the seminar, we will be using more than one access point because they typically support a fixed number of devices, either 5, 10, or 15 maximum connections. If 20-25 people powered up their CC3200 at the same time, they would all start "listening" for SmartConfig to provide the access point credentials. That would not be a problem if the access point could support all 20-25 connections, so some of those devices would not connect. So, how can we solve this problem? We could use the 128-bit AES encryption feature with SmartConfig. So that's what we need to modify in this application. The instructor will put you in groups, and let you know the access point credentials for your specific connection, along with the encryption key. You will then need to modify the `smartconfig.c` file in the project to use this encryption key.

- ▶ Open the `smartconfig.c` file so you can modify it.
- ▶ Find the `SmartConfigConnect()` function, and locate the call to `sl_WlanSmartConfigStart()` function.



```
62 void SmartConfigConnect()
63 {
64     unsigned char policyVal;
65
66     //set AUTO policy
67     sl_WlanPolicySet( SL_POLICY_CONNECTION,
68                     SL_CONNECTION_POLICY(1,0,0,0,0),
69                     &policyVal,
70                     1 /*PolicyValLen*/);
71
72     /* Start SmartConfig
73      * This example uses the unsecured SmartConfig method
74      */
75     sl_WlanSmartConfigStart(0, //groupIdBitmask
76                           SMART_CONFIG_CIPHER_NONE, //cipher
77                           0, //publicKeyLen
78                           0, //group1KeyLen
79                           0, //group2KeyLen
80                           NULL, //publicKey
81                           NULL, //group1Key
82                           NULL); //group2Key
83
84
85
86 }
```

**Note:** Obtain the publickey from your instructor. Each access point will use a different encryption key. This is the key you will also need to enter with SmartConfig. This should be a 16-character string. Easy to remember examples:  
 aaaabbbbccccdddd  
 bbbccccdddeeee

► Define the variable `publickey` and assign it your assigned 16-character encryption key. Find the following commented line in the `SmartConfigConnect()` function. Un-comment this line, and if necessary change the 16-character string.

```
const unsigned char publickey[16] = "aaaabbbbccccdddd";
```

► Comment out the current call to `sl_WlanSmartConfigStart()` function. Since this spans over several lines, you can highlight the lines, and then press Ctrl-/ (this has the effect of toggling comments).

► Now uncomment the following call. Highlight the block of code and press Ctrl-/

```
sl_WlanSmartConfigStart(1,           //groupIdBitmask
                        1,           //cipher
                        16,          //publicKeyLen
                        0,           //group1KeyLen
                        0,           //group2KeyLen
                        publickey,    //publicKey
                        NULL,         //group1Key
                        NULL);        //group2Key
```

Your code should be similar to this (your `publickey` may be different):


```
void SmartConfigConnect()
{
    const unsigned char publickey[16] = "aaaabbbbccccdddd";
    unsigned char policyVal;

    //set AUTO policy
    sl_WlanPolicySet( SL_POLICY_CONNECTION,
                     SL_CONNECTION_POLICY(1,0,0,0,0),
                     &policyVal,
                     1 /*PolicyValLen*/);

    /* Start SmartConfig
     * This example uses the unsecured SmartConfig method
     */
    // sl_WlanSmartConfigStart(0,           //groupIdBitmask
    //                          SMART_CONFIG_CIPHER_NONE, //cipher
    //                          0,           //publicKeyLen
    //                          0,           //group1KeyLen
    //                          0,           //group2KeyLen
    //                          NULL,        //publicKey
    //                          NULL,        //group1Key
    //                          NULL);       //group2Key

    sl_WlanSmartConfigStart(1,           //groupIdBitmask
                           1,           //cipher
                           16,          //publicKeyLen
                           0,           //group1KeyLen
                           0,           //group2KeyLen
                           publickey,    //publicKey
                           NULL,         //group1Key
                           NULL);        //group2Key
}
```


## 8. Re-build and run the thingfabric project

- Click the “bug”  icon on the toolbar to start the debugger

**Hint:** You’re now ready to run. Here is what to expect.

The code will start two TI-RTOS tasks. The first task is the out-of-box application task that connects to an access point. The second task is the `thingfabric` task which must be suspended, waiting for a “smart config done” semaphore to be signaled once the first task connects to the access point.

Right after you run the code, the red LED should blink several times. This indicates that the CC3200 is trying to find an access point in its list of profiles and if found, will attempt to connect. If you have never connected to this access point and there is no profile stored, the device will call the “smartconfig” function to go into smartconfig mode, where the device will just wait to be connected by the SimpleLink Wi-Fi Starter application. This is indicated by the red LED turning on solid (no blinking). The next step will be to run smartconfig from your phone/tablet, or by using another person’s phone/tablet to connect multiple devices.

- To run your code, click the Resume  button on the toolbar

Observe the behavior of the red LED, and get ready to connect the device to an access point using the SimpleLink Wi-Fi Starter application.

## Connect CC3200 to Access Point using SmartConfig

9. Start the SimpleLink Wi-Fi Starter application on your smartphone/tablet to connect your device (or a group of devices trying to connect to the same access point)

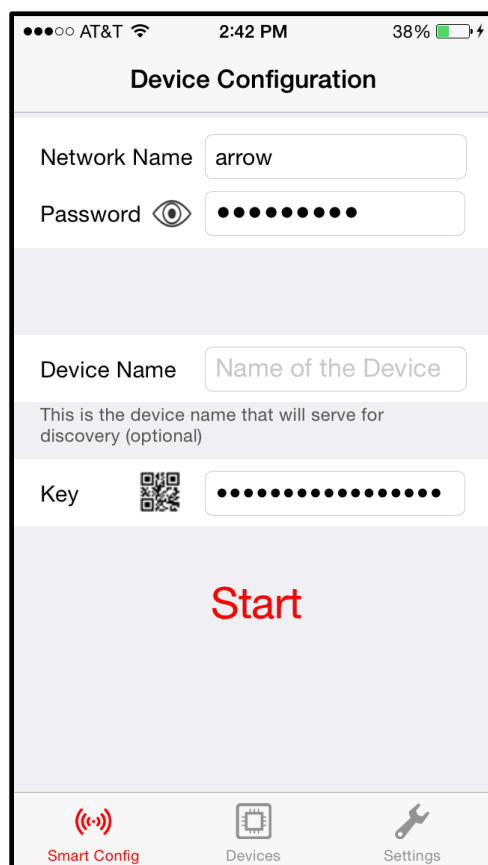
- ▶ Connect your smartphone/tablet to the access point
- ▶ Start the SimpleLink Wi-Fi Starter application on your smartphone/tablet
- ▶ If you don't see the "Key" field on the **SmartConfig** tab, click the **Settings** tab (on bottom of the screen) and select the option to "Show SmartConfig Key", then go back to the SmartConfig tab
- ▶ Enter the Network Name (should already be pre-filled with the SSID you are connected to)
- ▶ Enter the Password (usually supplied by your instructor)
- ▶ Enter the Key (usually supplied by your instructor)

---

**Note:** The Key you use here MUST match the value you entered in the application code you entered earlier.

---

- ▶ Press **Start**



The SmartConfig operation should complete in a few seconds. However, the operation can take up to two minutes to complete. The red LED should go off once connected.

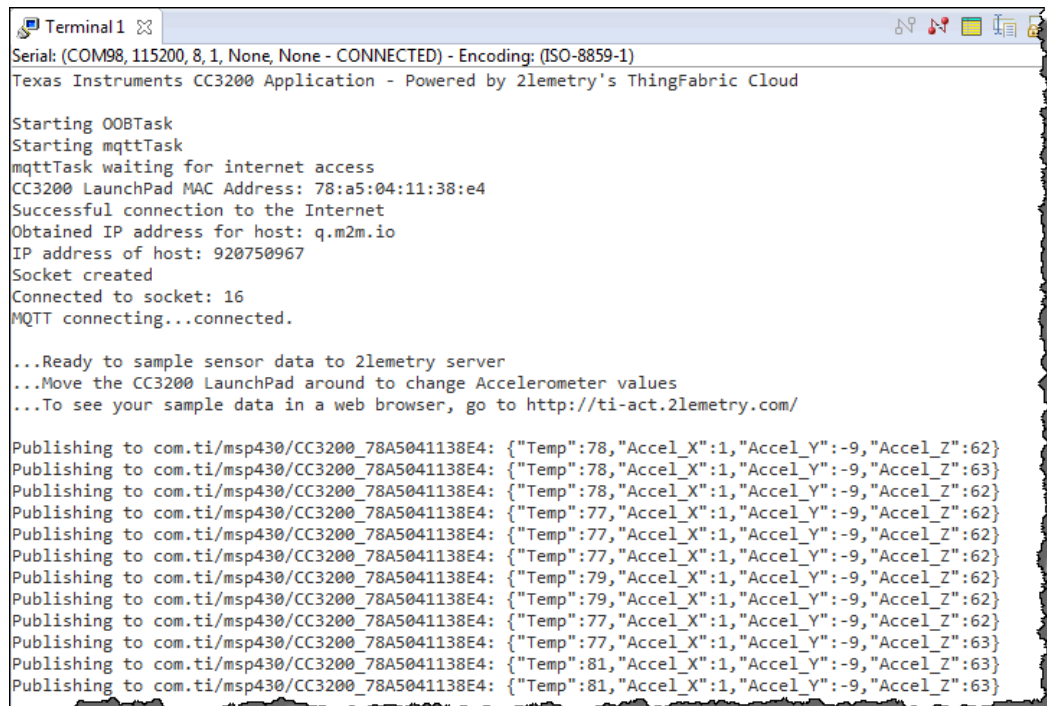


## Observe the terminal messages

### 10. You should see a number of status messages in the terminal window.

- From the terminal window, write down the MAC address for your device here:

You will need this later to distinguish your data values on the 2lemetry web portal.



```

Terminal1
Serial: (COM98, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
Texas Instruments CC3200 Application - Powered by 2lemetry's ThingFabric Cloud

Starting OOBTask
Starting mqttTask
mqttTask waiting for internet access
CC3200 LaunchPad MAC Address: 78:a5:04:11:38:e4
Successful connection to the Internet
Obtained IP address for host: q.m2m.io
IP address of host: 920750967
Socket created
Connected to socket: 16
MQTT connecting...connected.


...Ready to sample sensor data to 2lemetry server
...Move the CC3200 LaunchPad around to change Accelerometer values
...To see your sample data in a web browser, go to http://ti-act.2lemetry.com/

Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":78,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":78,"Accel_X":1,"Accel_Y":-9,"Accel_Z":63}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":78,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":77,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":77,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":77,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":79,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":79,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":77,"Accel_X":1,"Accel_Y":-9,"Accel_Z":62}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":77,"Accel_X":1,"Accel_Y":-9,"Accel_Z":63}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":81,"Accel_X":1,"Accel_Y":-9,"Accel_Z":63}
Publishing to com.ti/msp430/CC3200_78A5041138E4: {"Temp":81,"Accel_X":1,"Accel_Y":-9,"Accel_Z":63}
  
```

### 11. Once connected to the mqtt broker, the on-board sensor data will be sent to the 2lemetry mqtt broker about twice per second. You should see the red LED blink for each data set that is transmitted.

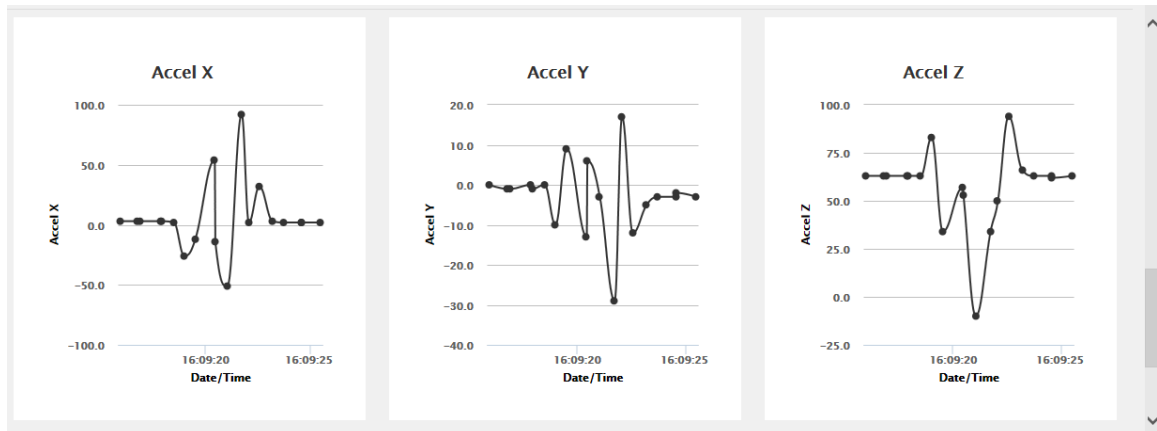
- Observe the temperature and accelerometer sensor values in the terminal window.
- To modify the sensor values, move the board around in three dimensions (this will affect the 3-axis accelerometer values). To modify the temperature value, hold your hand over the board, and then remove it.

### 12. Terminate the Debug session and close CCS

- On the toolbar click the Terminate button . This properly terminates the debug session, disconnects you from the target CC3200 LaunchPad board, and returns you to CCS edit mode.
- Close CCS

**13. Use your laptop to observe the sensor values you just collected on the 2lemetry web portal**

- Open your laptop's web browser and go to <http://ti-act.2lemetry.com/> It's best to use Chrome, FireFox, or IE v10 or greater.
- You should see a list of devices that connected. Using your MAC address (recorded in an earlier step in the lab), find your device and click on its name. You should see the values that were published to the cloud server.



That's it. You're done with this lab.

# Troubleshooting

## 1. Cannot connect to access point with SmartConfig

This is indicated by red LED staying lit solid. Check your encryption key in the code, and all the parameters to `sl_WlanSmartConfigStart()` function. Double check network name (SSID) and password. Verify with your instructor that the access point device limit has not been exceeded.

## 2. DNS error obtaining IP address for external site after connection to access point

If your access point has three DNS servers listed, remove one of the servers, and all should work o.k. – this is a known problem with one of the SimpleLink APIs to be fixed in an upcoming release. It can be found on the E2E forums.

## 3. Application gets stuck in an infinite loop inside a FreeRTOS “malloc” function

Make sure you’ve selected the TI-RTOS Default Build Configuration, re-build, and try again.

## 4. Trouble accessing the 2lemetry portal

Use Chrome, Firefox, or IE version 10 or above.

## 5. I don’t see data for Button 1, etc, on the 2lemetry web page

Portal shows no data for Button 1, Button 2, Joystick X, or Joystick Y. That’s normal since this portal was built for a different set of sensors. The only data being published is the accelerometer and temperature. There is no display for the temperature, although if you wanted to display the temperature data, you could send it to one of the other data values, say Joystick X – the label would still say it is Joystick data, but you know that you sent the temperature value there.

(blank page)

# The ThingFabric Portal Lab

---

## Introduction

The ThingFabric platform allows you to connect your devices and application clients using MQTT: a simple publish/subscribe protocol. For more information about MQTT please visit <http://mqtt.org> Our API allows you to send data via HTTP POST. Learn more about our API on [the portal](#).

The ThingFabric platform (MQTT Version 3.1.1 compliant at this time) allows incoming MQTT connections to q.thingfabric.com on port 1883. (If you're licensed for SSL, use port 8883). Any client connected is sandboxed within their "domain" level topic. Only clients with authorization are allowed to connect to the broker.

## Topics

<b>The ThingFabric Portal Lab .....</b>	<b>4-1</b>
<i>Important Terms</i> .....	4-3
MQTT .....	4-3
JSON.....	4-3
REST or RESTful .....	4-3
<i>Authentication</i> .....	4-5
Sign up for access to the portal .....	4-5
<i>Using the Portal</i> .....	4-6
Account .....	4-6
Projects .....	4-6
General.....	4-6
<i>Devices</i> .....	4-7
<i>Stream</i> .....	4-8
Publishing and Subscribing.....	4-8
Topic and Namespace .....	4-8
A little about the Payload .....	4-10
<i>Rules</i> .....	4-11
<i>Panels</i> .....	4-14
<i>Simulators</i> .....	4-17

# Important Terms

## MQTT

MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium. For more information about MQTT please visit <http://mqtt.org>

## JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

## REST or RESTful

REST is the underlying architectural principle of the web. The amazing thing about the web is the fact that clients (browsers) and servers can interact in complex ways without the client knowing anything beforehand about the server and the resources it hosts. The key constraint is that the server and client must both agree on the media used, which in the case of the web is HTML.

An API that adheres to the principles of REST does not require the client to know anything about the structure of the API. Rather, the server needs to provide whatever information the client needs to interact with the service. An HTML form is an example of this: The server specifies the location of the resource, and the required fields. The browser doesn't know in advance where to submit the information, and it doesn't know in advance what information to submit. Both forms of information are entirely supplied by the server.

So, how does this apply to HTTP, and how can it be implemented in practice? HTTP is oriented around verbs and resources. The two verbs in mainstream usage are GET and POST, which I

think everyone will recognize. However, the HTTP standard defines several others such as PUT and DELETE. These verbs are then applied to resources, according to the instructions provided by the server.

---

**Note:** **Domain** is synonymous (or interchangeable) with **Project**  
**Group** is synonymous (or interchangeable) with **Stuff**  
**Thing** is synonymous (or interchangeable) with **Device**

---

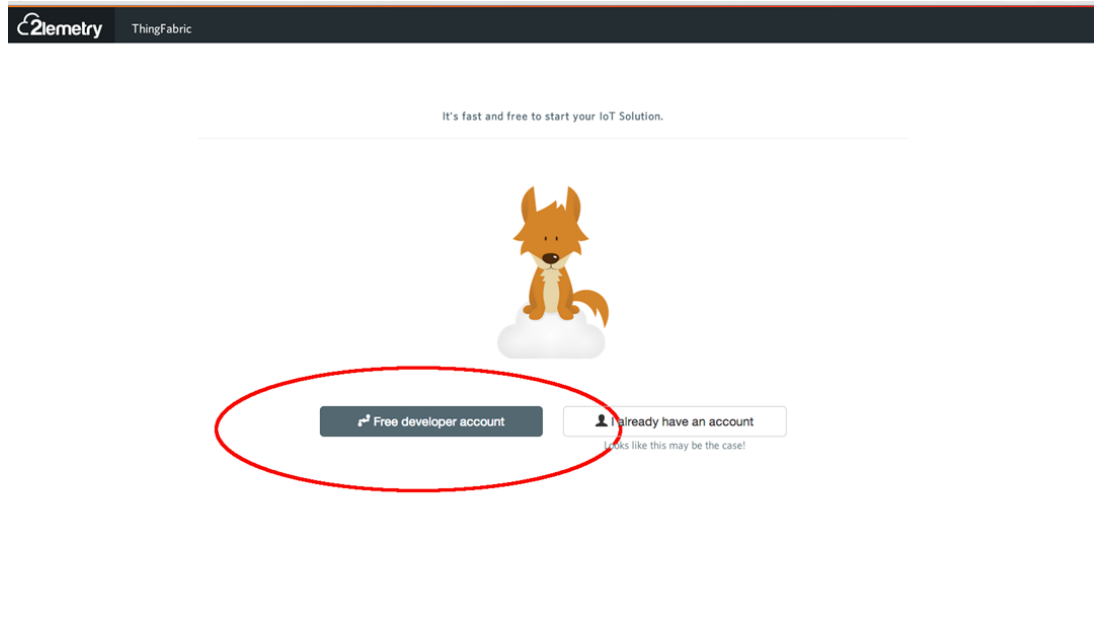


# Authentication

All clients connecting to the platform require a username and password. You can sign up for an account at [app.thingfabric.com](http://app.thingfabric.com)

## Sign up for access to the portal

If this is your first time using the portal, [sign up here](#). You can also link other accounts here, such as Github, Google or [Salesforce.com](https://www.salesforce.com) and use them to log in.



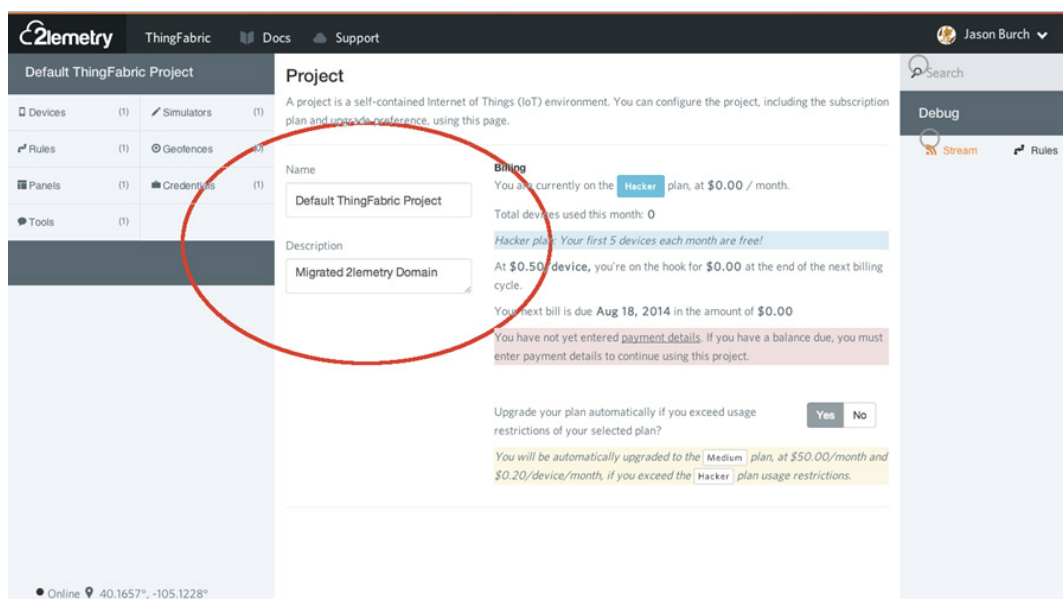
## Using the Portal

### Account

You can see your Account Settings by clicking on your name at the top right. You can merge other accounts here such as Github, Google and [Salesforce.com](https://www.salesforce.com)

### Projects

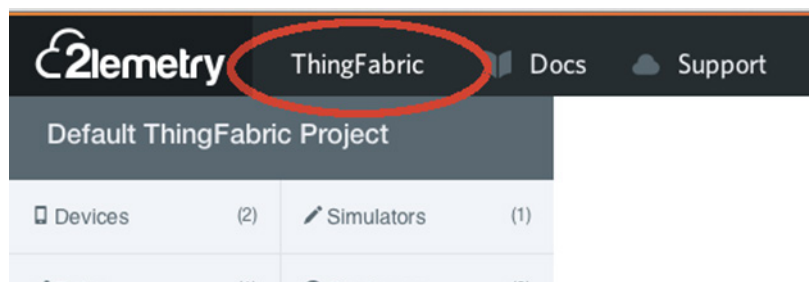
After signup you can log in to see your Projects. A default project is assigned automatically. By clicking on your project you can **name the project and give it a description**. This is required when publishing or subscribing to data. Domains are automatically provisioned for new accounts.



You can also see which users have access to the project and invite new users to join a project. If you know the users ThingFabric ID, you can use that to invite them to your project, otherwise you can use their email address.

### General

You can always return to the Project menu by clicking ThingFabric at the top of the page.

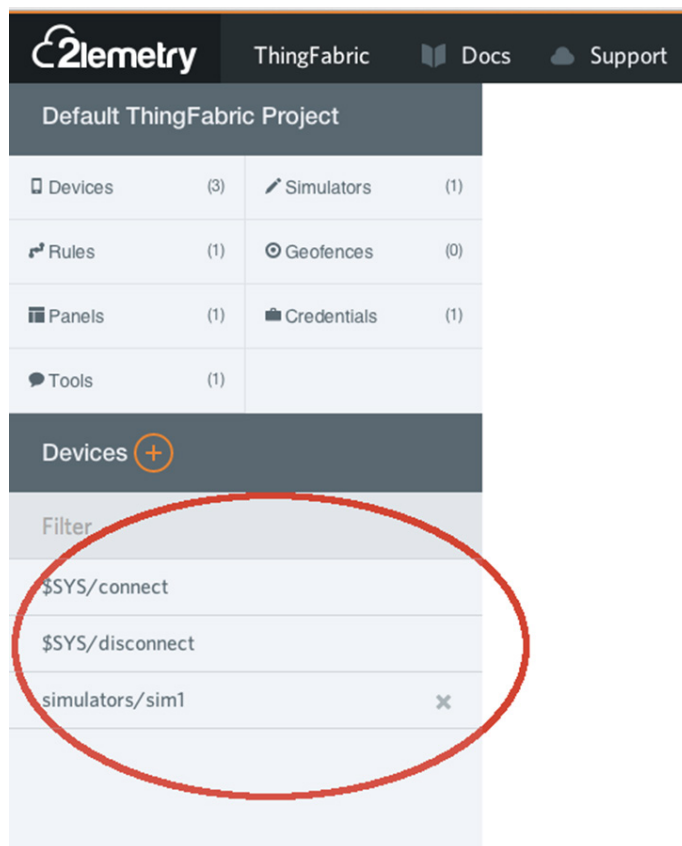


## Devices

If you choose the Devices tab from the Project Menu you will see anything that is 3rd level on a 3 level topic. project/stuff/thing

These can even be system generated in the case of project/\$SYS/connect and project/\$SYS/disconnect

There is also a simulator example that has been created for you.



## Stream

You can use our webclient to simulate data that a device would send using a topic and payload. You can also send data to an actual device if that device is subscribed to a topic.

Or you can Download a compatible 3.1 mqtt client. We recommend the Eclipse Paho clients for Java or C. The source can be downloaded from <https://github.com/eclipse/paho.mqtt.java> or <https://github.com/eclipse/paho.mqtt.c>

When using these clients remember to md5 hash your password.

Choose **Tools** from the project menu and you will see our mqtt web client. This allows you to publish and subscribe to topics and easily interact with devices or simulated data. You'll see something like this. Connect.

The screenshot displays the MQTT Client web interface. It features a 'Subscribe' section on the left with a 'Topic' input field containing '10b27232eb7c6fd8770b7cac95400f52/#' and a green 'Subscribe' button. Below this is a 'Subscriptions' section showing 'No active subscriptions yet.' On the right, the 'Publish' section has a 'Topic' input field with '10b27232eb7c6fd8770b7cac95400f52/hello', a 'Message' input field with 'Your MQTT message.', and a green 'Publish' button. At the bottom, a 'Messages' section shows 'No messages were received yet.' The interface includes a top navigation bar with 'MQTT Client', 'Search', 'Debug', 'Stream', and 'Rules' tabs. The 'Stream' tab is currently active.

## Publishing and Subscribing

### Topic and Namespace

Topic is typically a 3 level string separated by forward slashes.

We generally use and recommend a structure that looks like this: <project>/<stuff>/<thing>

You can have more OR less than 3 levels however we only store 3 level data by default. If you have a use case for more or less let us know, however we find this works for most cases. By default the topic namespace for a platform client must start with the users 'project'. The next level of the namespace, 'stuff' is a logical grouping of related things. The third level of the namespace, 'thing' relates to the client's unique identifier.

If you don't follow the 3 level namespace pattern described above, your messages will still be passed by the broker to subscribers as long as the users have rights to publish/subscribe on the given topic.

Note: messages will not be stored if the topic space is different.

Subscribing is also based on a TOPIC string.

For my example I will use **10b27232eb7c6fd8770b7cac95400f52/CC3200/device1**

The screenshot shows a web-based MQTT client interface. On the left, under the 'Subscribe' tab, the 'Topic' field contains the string '10b27232eb7c6fd8770b7cac95400f52/CC3200/de'. Below this is a green 'Subscribe' button. At the bottom left, a 'Subscriptions' list shows the same topic string with a green 'II' (pause) button and a red 'X' (unsubscribe) button. On the right, under the 'Publish' tab, the 'Topic' field also contains the same string. Below it, the 'Message' field contains the JSON object '{"temp":100}'. A green 'Publish' button is located at the bottom right.

Now Publish on the same topic that you are subscribed to and you will see the message on the right. Notice how quickly this transaction takes place. If this data is sent in proper JSON format it will be automatically stored and can be retrieved using our API. You can check formatting with online tools like <http://jsonlint.com/>

Publish on **<project>/CC3200/device2>** and note the message is not received because the client is only listening for data coming in on /device1 . We can listen for all devices within a grouping by using the wildcard #

So, subscribing to **<project>/CC3200/#** will capture all data related devices in my <project> and also within my group CC3200. Additionally we can capture data from all devices in our domain by subscribing to <project>/#

Now I can publish to device1, device1000 or device1000000 and I will get the message!

Try a few scenarios. This will help you get a better understanding of how mqtt, publishing and subscribing works.

Now if the data was published in proper JSON it will be automatically stored. We can onboard custom protocols or translate binary, xml or whatever to JSON, but that needs to be pre-defined for a given customer.

## A little about the Payload

The payload data must be in JSON object format to be parsed and saved to the datastore.

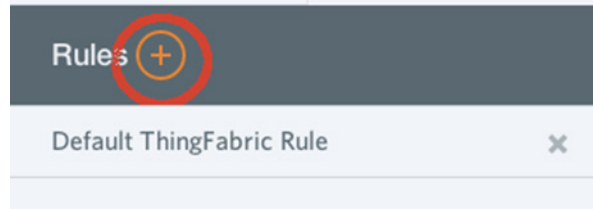
Any single or nested JSON object is allowed. All other topic patterns or payload structure would not be stored but are allowed for pub/sub. An example would be to publish to a domain like this <project>/mystuff/device123 and then of course to subscribe on the same topic to get the data from that device only. Additionally you can subscribe on <project>/mystuff/# to get data from all the devices in the mystuff category, or <project>/# to get data on all categories and all devices in that domain. We can accept any payload, XML for example, but it won't be stored. Messages will still be passed to subscribers and over websockets. This is an example of a valid payload that would be stored in the platform datastore:

```
{
  "report": {
    "gps": {
      "latitude": "39.702610038220882",
      "longitude": "-104.97231300920248"
    },
    "temp": {
      "celsius": "252",
      "fahrenheit": "76"
    }
  }
}
```

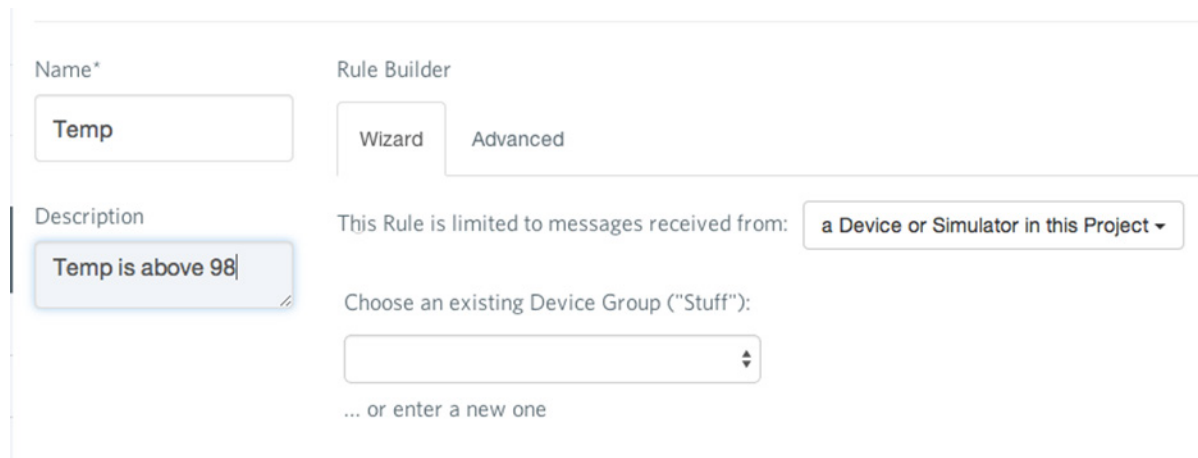
## Rules

Creating rules inside the portal is very easy. First click the rules button in the project menu.

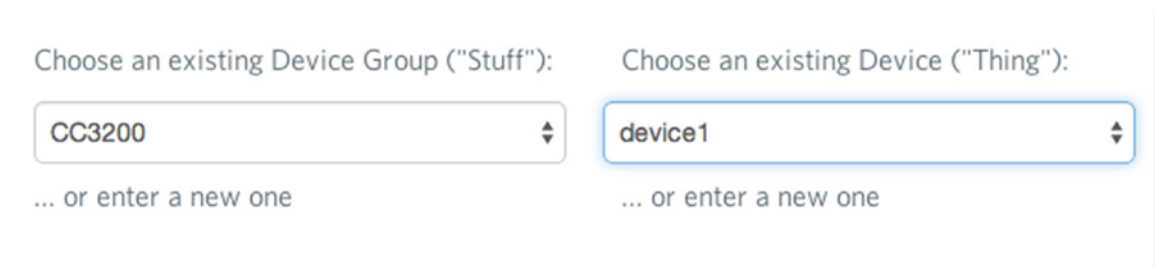
Now you will see a list of your rules (note: an example has been created for you) and an Add Rule button. Click here to begin creating a rule.



After clicking Add Rule, give your rule a Name and Description. Next, choose 'a Device or Simulator in this Project' from the drop-down list.



Next choose the Device Group you want to filter on. Here we are going to pick CC3200. It is a known Group because we have published data to this group already. We are going to choose device1. Again, it already exists in our device list for this group because we published data using this device in the Stream.



Next we will select 'only when specific data is received' from the drop down. Note you can also fire a rule anytime data is received from this device regardless of the data.

If a message is received from the Device **device1** in the Device Group **CC3200**

... then the rule should fire:

only when specific data is received ▾

regardless of what data is received

only when specific data is received

temp

50

Next you need to select the marker next to temp in order to enter the info



100

We are going to choose > 'greater than' and leave the default value there of 100.

temp > 100



Finally you choose the Action you want to take place if the rule is met. These actions include:

- Sending an Email
- Sending an SMS
- Pushing the data via HTTP
- Send a message
- Resend a message

Let's look at sending an sms. For SMS the phone number should be in format country code plus number. Within the US 15551231234. Click the Add button.

... and invoke these actions:

0 Send E-mail

0 Send SMS

0 HTTP POST

0 Send Message

0 Resend Message

To

17203529039

Message

temp to hot

Add

You will get a confirmation that the rule has been saved.

Next lets fire the rule using the Stream tool. Click Tools. Change the publish to include CC3200/device1 after the project id. Message {"temp":101} Should look something like this. Publish and get your SMS

Publish

Topic

232eb7c6fd8770b7cac95400f52/CC3200/device1

Message

{"temp":101}

Publish

## Panels

Panels are widgets that allow you to visualize your data. When you choose Panels from the left side of the screen you will see a list of your panels (*note: an example has been created for you*), and an Add Panel button. Click here to begin creating a panel. Give your panel a name and a description. Lets leave the Panel set to No under Public.

The Panel Editor form consists of three input fields on the left and a list of widgets on the right. The 'Name\*' field contains 'Temp'. The 'Description' field contains 'Temperature'. The 'Public\*' field has two buttons: 'Yes' and 'No', with 'No' being the selected option. To the right, under the heading 'Panel Editor', there are two widget cards. The first card is titled 'Table Stream (Devices)' and features a Wi-Fi icon. The second card is titled 'Table Past (Device)' and features a bar chart icon.

Drag Table Stream (Devices) block to a Widget Slot free location. And drag Table Past to an available location

This screenshot shows the Panel Editor with the 'Table Stream (Devices)' and 'Table Past (Device)' widgets on the left. On the right, there are four widget slots, each labeled 'Widget slot free' in blue text. The 'Table Stream (Devices)' widget is being dragged into the top-left slot, and the 'Table Past (Device)' widget is being dragged into the bottom-left slot.

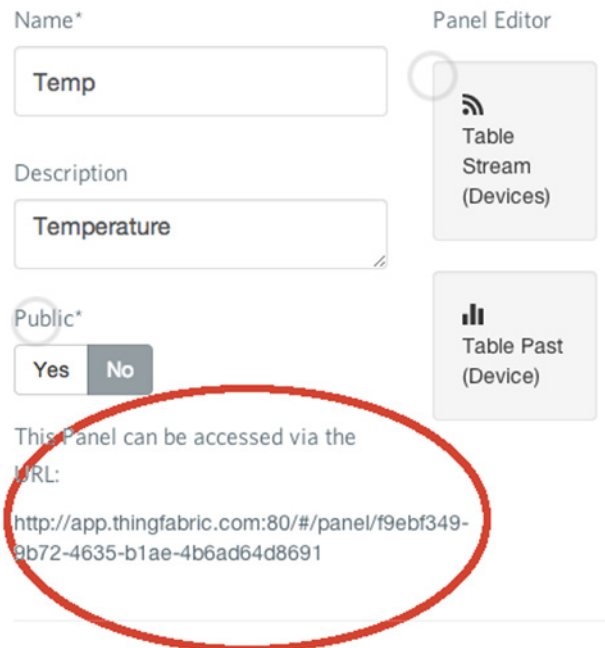
Next you see the Widget Editor. Give the Widget a Name and a Credential.

Choose a Device or Simulator in this Project

Choose CC3200

Choose device1

Click the Save Button in Widget Editor and click the Save Button in the Panel Note you have a few locations available for other Widgets. You get a notification that the Panel has been created. Note this Panel can be accessed via the URL:



Name\*

Temp

Description

Temperature

Public\*

Yes No

This Panel can be accessed via the URL:

<http://app.thingfabric.com:80/#/panel/f9ebf349-9b72-4635-b1ae-4b6ad64d8691>

Panel Editor

Table Stream (Devices)

Table Past (Device)

Right click the link and open in a new tab. Next go to Tools and we will publish some data to the Panel. Click Tools. Change the publish to include CC3200/device1 after the project id. Message {"temp":101} Should look something like this.

Publish

Topic

232eb7c6fd8770b7cac95400f52/CC3200/device1

Message

{"temp":101}

Publish

Publish and go to your Panel URL see data.

**Temp**  
Temperature

Timestamp

Message

Jul 21, 2014 11:54:20 AM	<div>("temp":150)</div> <div>Expand</div>
--------------------------	-------------------------------------------

10b27232eb7c6fd8770b7cac95400f52/CC3200/device1/#

07/21/2014

07/21/2014

Update

Timestamp

Message

Jul 21, 2014 11:42:50 AM	<div>{{"temp": "98"}}</div> <div>Expand</div>
Jul 21, 2014 11:01:52 AM	<div>{{"temp": "101"}}</div> <div>Expand</div>
Jul 18, 2014 3:38:50 PM	<div>{{"temp": "100"}}</div> <div>Expand</div>

## Simulators

Simulators can be used to create Device traffic without having a physical piece of hardware. Simulator data can be created with the payload builder, or JSON can be pasted in. Note that the simulator will run even when you navigate away from this page or log out!

Click Simulators. Now you will see a list of your simulators (*note: an example has been created for you*), and an Add Simulator button. Click here to begin creating a simulator.



Give the simulator a

- Name
- Description
- Credential
- Stuff
- Thing
- Frequency

Name\*

Temp

Description

temp

Credential\*

1a02d796-fcea-4bbf-918f-62t ⬆ ⬇ ⬇ ⬆

Stuff\*

CC3200

Thing\*

device1

Frequency (in seconds)\*

10



Go to the Advanced tab and enter {"temp":100}

Message Builder

☐ Wizard ☒ Advanced

Paste your JSON message here if you can't build it with the Wizard!

```
{"temp":100}
```

Click Save. Now go into your Simulator and click Start. Open your Panel and see the data flow.

(blank page)



# Personalize ThingFabric Application Lab

---

## Introduction

In the previous ThingFabric lab, we collected sensor data from the CC3200 LaunchPad and published this data to the 2lemetry mqtt broker. However, in that application, we were publishing to a common "TI/Arrow seminar account". Since you have now created your own personal account, you can modify the application to publish the data to your own account.

## Lab Objectives

- Modify the log-in credentials in the ThingFabric project to connect the device to your own account
- Use an online MD5 hash utility to generate the proper password
- Run the application to observe the device publishing mqtt messages to your account
- Display the temperature values being published from your CC3200 LaunchPad device

## Chapter Topics

<b>Personalize ThingFabric Application Lab .....</b>	<b>5-1</b>
<i>Preparation Checklist .....</i>	<i>5-3</i>
<i>Modify the ThingFabric Code and Run Application .....</i>	<i>5-4</i>

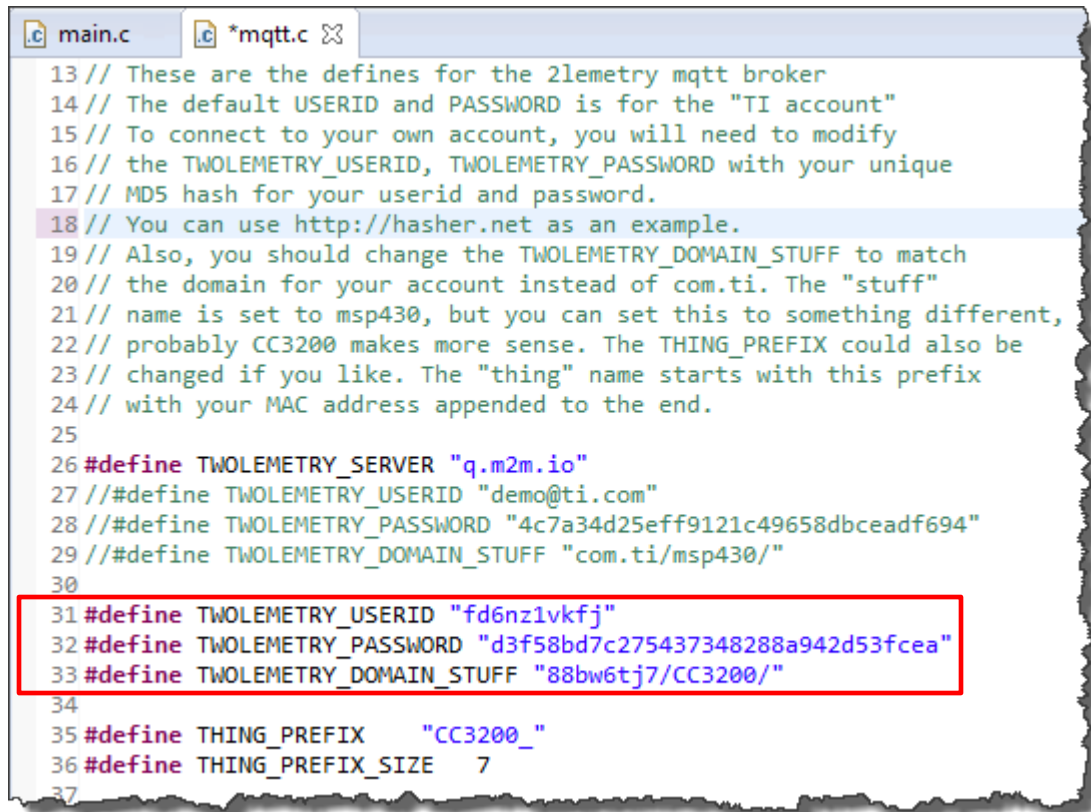
## Preparation Checklist

- ☐ Make sure the SOP2 jumper is connected on your LaunchPad board
- ☐ Verify there is no jumper on VCC-P58
- ☐ Connect your LaunchPad board to an available USB port on your computer
- ☐ Open Code Composer Studio (CCS)
- ☐ Select the thingfabric project in Project Explorer to make it the [Active] project
- ☐ Ensure that TI-RTOS build configuration is selected

## Modify the ThingFabric Code and Run Application

### 1. Modify the #define constants in the application to connect to your account

- ▶ Open the `mqtt.c` file so you can modify it.
- ▶ Near the top of the file `mqtt.c`, locate the #define constants for:  
    `TWOLEMETRY_USERID`,  
    `TWOLEMETRY_PASSWORD`, and  
    `TWOLEMETRY_DOMAIN_STUFF`



```
13 // These are the defines for the 2lemetry mqtt broker
14 // The default USERID and PASSWORD is for the "TI account"
15 // To connect to your own account, you will need to modify
16 // the TWOLEMETRY_USERID, TWOLEMETRY_PASSWORD with your unique
17 // MD5 hash for your userid and password.
18 // You can use http://hasher.net as an example.
19 // Also, you should change the TWOLEMETRY_DOMAIN_STUFF to match
20 // the domain for your account instead of com.ti. The "stuff"
21 // name is set to msp430, but you can set this to something different,
22 // probably CC3200 makes more sense. The THING_PREFIX could also be
23 // changed if you like. The "thing" name starts with this prefix
24 // with your MAC address appended to the end.
25
26 #define TWOLEMETRY_SERVER "q.m2m.io"
27 // #define TWOLEMETRY_USERID "demo@ti.com"
28 // #define TWOLEMETRY_PASSWORD "4c7a34d25eff9121c49658dbceadf694"
29 // #define TWOLEMETRY_DOMAIN_STUFF "com.ti/msp430/"
30
31 #define TWOLEMETRY_USERID "fd6nz1vkfj"
32 #define TWOLEMETRY_PASSWORD "d3f58bd7c275437348288a942d53fcea"
33 #define TWOLEMETRY_DOMAIN_STUFF "88bw6tj7/CC3200/"
34
35 #define THING_PREFIX "CC3200_"
36 #define THING_PREFIX_SIZE 7
37
```

These are the three lines that need to be modified to connect to your account.

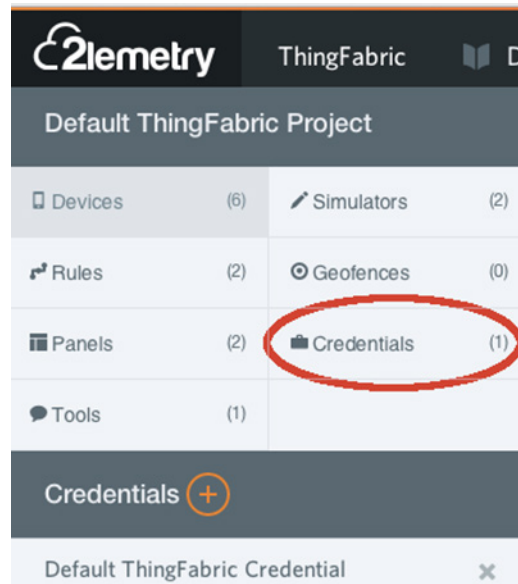
---

**Note:** Don't use the values shown here. You need to find your own account login credentials to make these modifications to the source code. That is the next step in the lab.

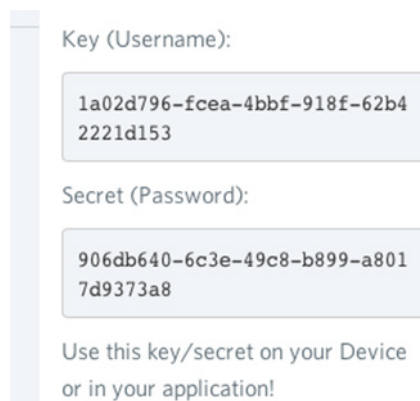
---

## 2. Log in to your account at <https://app.thingfabric.com>

- ▶ If you have not created an account you can do this now
- ▶ Once logged into your account, click on one of your projects and click the “Credentials” button on the upper left of the screen



- ▶ Under “Your Credentials”, click “Default”



- ▶ Write down the value for **Key (Username)**: \_\_\_\_\_

This is the userid value that you will need to log in to the broker

- ▶ Write down the value for **Secret (Password)**: \_\_\_\_\_

This is the value you need for the password. But for this value, you need to use the token's MD5 hash value.

## 3. Obtain the MD5 hash value for your password (Token)

- ▶ Using your web browser, go to <http://md5hasher.net> to get the MD5 hash value for your password. It's probably easiest to copy your password using Ctrl-C on the thingfabric page, then paste it into the md5hasher page. Click the “Hash It ->” button.

**4. Copy/Paste your Username value**

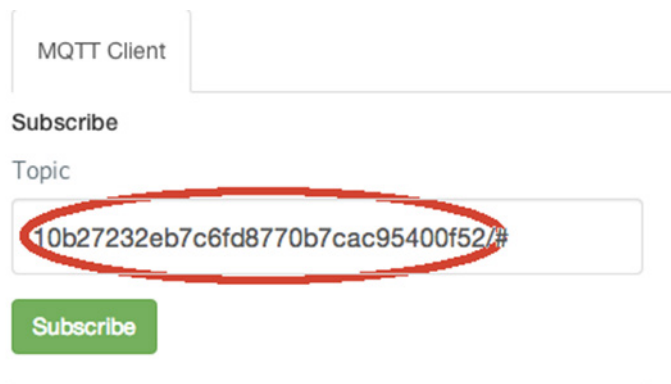
- ▶ Copy (Ctrl-C) and Paste (Ctrl-V) your Username value to the  
`#define TWOLEMETRY_USERID "Username value"`

**5. Copy/Paste the MD5 value for your password (Token)**

- ▶ Copy (Ctrl-C) and Paste (Ctrl-V) this hash value to the  
`#define TWOLEMETRY_PASSWORD "Token MD5 hash value"`  
The actual "Token MD5 hash value" will be unique to your Token credential.

**6. Copy/Paste your project name**


- ▶ Copy (Ctrl-C) and Paste (Ctrl-V) your project name to  
`#define TWOLEMETRY_DOMAIN_STUFF "xxxxxxxxxx/CC3200/"`



The screenshot shows a web interface for an MQTT Client. At the top, there is a label 'MQTT Client' and a 'Subscribe' button. Below this, there is a 'Topic' label and a text input field. The input field contains the text '10b27232eb7c6fd8770b7cac95400f52/#', which is circled in red. Below the input field is another green 'Subscribe' button.

Your Project name will be different, of course. The "stuff" name CC3200 is arbitrary, recall that it is the next level below your Domain, used to group devices (things).

**7. Re-build and Load the Application**

- ▶ Click the "bug"  icon in the toolbar to build the project and start the debugger
- ▶ If you encounter any build errors, examine them, fix the problems, and re-build

**8. At the thingfabric portal, Subscribe to the topic being published by the LaunchPad board so you can see the values there**

- Before you run your code, you should go the 2lemetry web portal and click on Tools

Subscribe

Topic

10b27232eb7c6fd8770b7cac95400f52/#

Subscribe

- Click the “Subscribe” button, and you will see the subscription. You can PAUSE the subscription or cancel it.

**9. Run your code and observe the live data being published**

- To run your code, click the Resume  button on the toolbar

Your device should connect to the same access point you connected to during a previous lab. How does the board know to connect to that access point? After you got a successful connection using SmartConfig, the profile for the access point was stored by the application into the serial flash. The device will always try to connect to the access point(s) in the stored profile(s).

- Observe status of your session in the terminal window. If your account credentials were done correctly, you should see this in the terminal window:


```
MQTT connecting...connected.
```

If you do NOT get this message, make sure to check that the Username and MD5 Password strings are correct in your code

► You should also see the data values being posted continuously, every 500 milliseconds, for example:

The screenshot displays the MQTT Client interface. At the top, there's a tab labeled 'MQTT Client'. Below it, the interface is divided into two main sections: 'Subscribe' and 'Publish'.  
In the 'Subscribe' section, the 'Topic' field contains the value '10b27232eb7c6fd8770b7cac95400f52/#'. Below this is a green 'Subscribe' button. Underneath the 'Subscribe' button is a 'Subscriptions' list. This list contains one entry: '10b27232eb7c6fd8770b7cac95400f52/#', which has a green 'II' (pause) button and a green 'X' (close) button next to it.  
In the 'Publish' section, the 'Topic' field contains the value '10b27232eb7c6fd8770b7cac95400f52/hello'. Below this is a 'Message' input field with the placeholder text 'Your MQTT message.' and a green 'Publish' button.  
At the bottom of the interface is a 'Messages' section. It shows a list of received messages. The first message is highlighted and its details are expanded. The message ID is '10b27232eb7c6fd8770b7cac95400f52/CC3200/CC3200\_78A50411054B' and the QoS is '0'. The message payload is a JSON object: `{"Temp":89,"Accel_X":3,"Accel_Y":-3,"Accel_Z":63}`. Below the message list is a green 'Expand' button.

## 10. Terminate the Debug session

► On the toolbar click the “Terminate” icon . This properly terminates the debug session, disconnects you from the target CC3200 LaunchPad board, and returns you to CCS edit perspective.

## 11. Close CCS