

# SimpleLink SDK Wi-Fi Plugin Users Guide

## Table of Contents

---

- [Introduction](#)
- [Plugin Architecture / General Overview](#)
- [Folder Structure](#)
- [Updating the CC31xx \(Service Pack and User Files\)](#)
  - [Uniflash / CC31XXEMU-BOOST](#)
  - [Over-the-Air Updates](#)
- [Mobile Applications](#)
  - [SimpleLink SDK Starter Pro](#)
- [Serial Port Logging/Debugging](#)
- [Customizing Parameters](#)
- [Code Examples](#)
- [IDE/Software Configurations](#)
  - [Code Composer Studio Project Guide](#)
  - [IAR Embedded Workbench Project Guide](#)
- [Rebuilding the SimpleLink Wi-Fi Host Driver](#)
  - [Prerequisites](#)
  - [Building with CCS](#)
  - [Building with GCC and IAR](#)
- [Support](#)

## Introduction

The SimpleLink™ SDK Wi-Fi Plugin is a companion software package that enables the use of a Wi-Fi radio on any standard MSP432 platform including the [MSP432P401](#), [MSP432P4111](#), and [MSP432E401](#) families, as well as on the CC26X2R platform, including the [CC2642R](#) and [CC2652R](#) family of devices. By having the ability to seamlessly and modularly add Wi-Fi functionality to an embedded system, a programmer can enable their embedded device to become a gateway to various IOT infrastructures.

This plugin leverages the use of the Wi-Fi host driver that is standard within the [SimpleLink CC32xx SDK](#). Through a simple porting layer interaction with the network processor (NWP) is abstracted out to enable code compatibility between a one chip CC32xx solution and a two chip host microcontroller + CC31xx solution. A SPI connection is used by the host microcontroller to manipulate and configure all aspects of the Wi-Fi network processor.

SimpleLink SDK Plugins are designed to work in tandem to the microprocessor specific SDK. This plugin is designed to work alongside both the the [SimpleLink MSP432P4 SDK](#), [SimpleLink MSP432E4 SDK](#), and [SimpleLink CC26X2 SDK](#). It relies heavily on several core elements within these SDKs. Without a prior installation of the SimpleLink SDK, the SimpleLink SDK Wi-Fi Plugin will not work. While testing of this software package was performed on release 2.30.00.xx of the SimpleLink SDKs, it is likely that it will also work with newer and older versions.

## Plugin Architecture / General Overview

All of the code examples written for this software package leverage a combination of various host controller LaunchPads. The supported host microcontroller LaunchPad configurations are listed below:

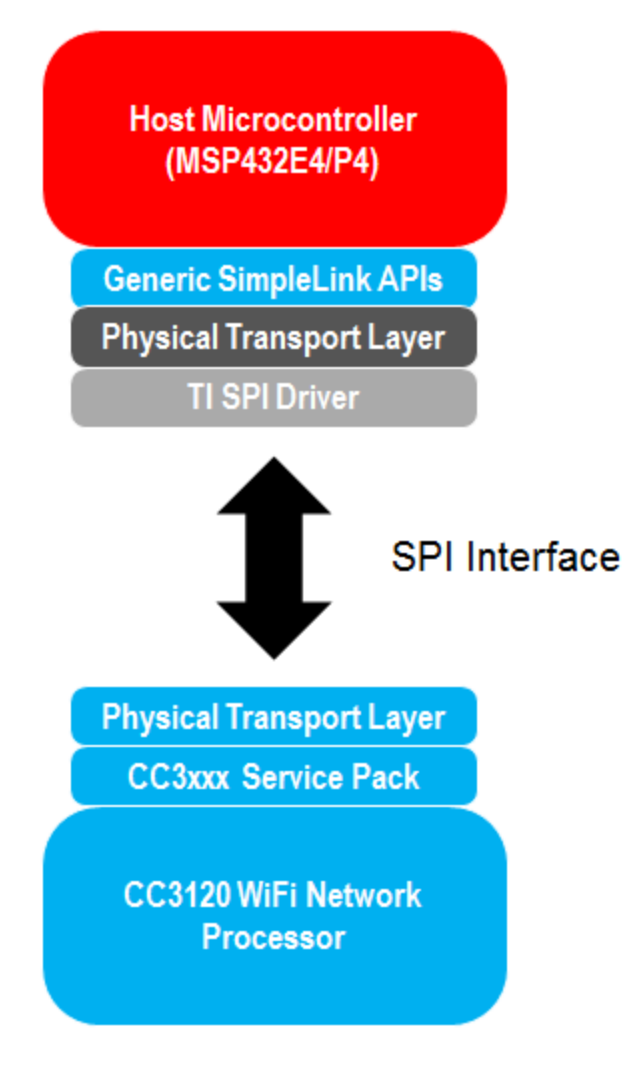
- [MSP-EXP432P401R](#)
- [MSP-EXP432P4111](#)
- [MSP-EXP432E401Y](#)
- [LAUNCHXL-CC26X2R1](#)

The network processors that can be used in this plugin as the main vessel of communication are listed below:

- [CC3120 BoosterPack](#)
- [CC3135 BoosterPack](#)

A BoosterPack is used in conjunction with the host LaunchPad to provide the benefits of a full capability host microcontroller coupled with the modularity of a separate network processor for Wi-Fi communication.

The communication between the host and the C31xx has been abstracted away using a simple porting layer that sits above the Wi-Fi host driver on the host microcontroller. While the underlying host driver is identical to what you would see in the CC32xx SDK, the porting layer translate all of the high level Wi-Fi calls to a physical SPI layer that communicates with the CC31xx as a black box solution. This allows for the programmer to call SimpleLink Wi-Fi APIs (such as *sl\_Start*, *sl\_Fs*, etc.) calls as they would on a CC3220. A brief block diagram of the various software connections and driver interworkings can be seen below:



The physical interface between the host and the CC31xx is a simple SPI connection with two interrupt lines for synchronization and power management added. The physical hardware specification and interworkings are explained in detail in the [CC3120 BoosterPack User's Guide](#). It is important to note that while the default pin configurations used in the example projects in this plugin package are mapped to the LaunchPad pins, they are fully reconfigurable to match the end application requirements. Reconfiguring the pin configurations and SPI driver parameters is explained in detail in the [Customizing Parameters](#) section of this document.

## Folder Structure

The folder structure for the SimpleLink SDK Wi-Fi Plugin is made to compliment the standard folder structure that the platform SDK adheres to. From the root directory, the following folders are available:

- **.metadata/** - Internal folder for product integration into Code Composer Studio. You do not have to edit any files in this folder.
- **docs/** - Contains all documentation related to the SDK plugin
- **kernel/** - Contains any custom RTOS kernel configuration files
- **examples/** - Contains all of the [code examples](#) and their IDE project files

- **source/** - Contains generic/reusable code that can be imported into end applications
- **tools/** - Contains various tools/utilities that supplement the release

The *source/* folder contains the code that is likely to end up seeing its way into an end application. Within there is the following folders:

- **ti/** - TI provided collateral
- **third\_party/** - Third party provided collateral

The *ti/* folder contains the bulk to the TI provided collateral to interact with and control the CC31xx Wi-Fi Device. This folder is structured as follows:

- **drivers/net/wifi** - Contains the host driver implementation for communication between the host MCU and the CC31xx over SPI
- **net/json** - Source and pre-built libraries for parsing JSON files (catered for over-the-air updates)
- **net/mqtt** - Driver implementation for the [MQTT connectivity protocol](#)
- **net/ota** - Over-the-air firmware update libraries. Used to update user files/service packs on the CC31xx as well as the firmware on the MSP432

The entirety of code provided by TI is provided as an open source BSD license and generally include both prebuilt static libraries for each supported IDE as well as full source code. The *ota* folder does not contain prebuilt libraries as much of the ota implementation is dependent on end user requirements and parameters.

The *third\_party/* folder contains the [mbedtls](#) library that is used by the MSP432 for SHA256 generation in lieu of a hardware SHA256 module (that the CC3220 contains).

Back in the root directory, the *tools/* folder consists of the following two folders:

- **iar/** - Metadata needed to enable functionality in IAR Embedded Workbench
- **cc31xx\_tools/** - Miscellaneous collateral and tools required for device operation

The *cc31xx\_tools/* directory contains various collateral needed to enable functionality of various functions. This includes the following subdirectories:

- **certificate-catalog/** - Used for signing and securing OTA firmware updates
- **certificate-playground/** - Miscellaneous dummy certificates used for development and evaluation
- **ota-example-cert/** - Dummy signing certificates used for SHA256 calculation for OTA firmware updates
- **servicepack-cc3x20/** - Service pack of Wi-Fi stack running on CC3120
- **servicepack-cc3x35/** - Service pack of Wi-Fi stack running on CC3135

## Updating the CC31xx (Service Pack and User Files)

It will become necessary for the user to update both the user files stored on the CC31xx's flash storage and the service pack running on the CC31xx itself. User files that need to be updated can range from specific

application-centric files required for operation (such as html pages) to updated certificates and trusted root certificate files.

The service pack updates are done on a regular basis to patch fixes for the services running on the NWP as well as to ensure that the NWP is running with the most up-to-date security patches. It is the same process to program the CC3120 and the CC3135, with the only differences being that the user needs to ensure that the correct service pack and Device Type are chosen for the specific device.

The CC3120 and the CC3135 use **different servicepacks**, but the **same user files**.

The service pack for the CC3120 is located in the plugin in *tools/cc31xx\_tools/servicepack-cc3x20*. This is identical to the service pack distributed in the CC32xx SDK in *tools/cc32xx\_tools/servicepack-cc3x20*.

The service pack for the CC3135 is located in the plugin in *tools/cc31xx\_tools/servicepack-cc3x35*. This is identical to the service pack distributed in the CC32xx SDK in *tools/cc32xx\_tools/servicepack-cc3x35*.

The sections below describe not only how to update the service pack and user files running on the CC31xx via a direct USB connection, but also how to use over-the-air updates (OTA) to dynamically load a customized update package via the device's Wi-Fi connection.

## Uniflash / CC31XXEMU-BOOST

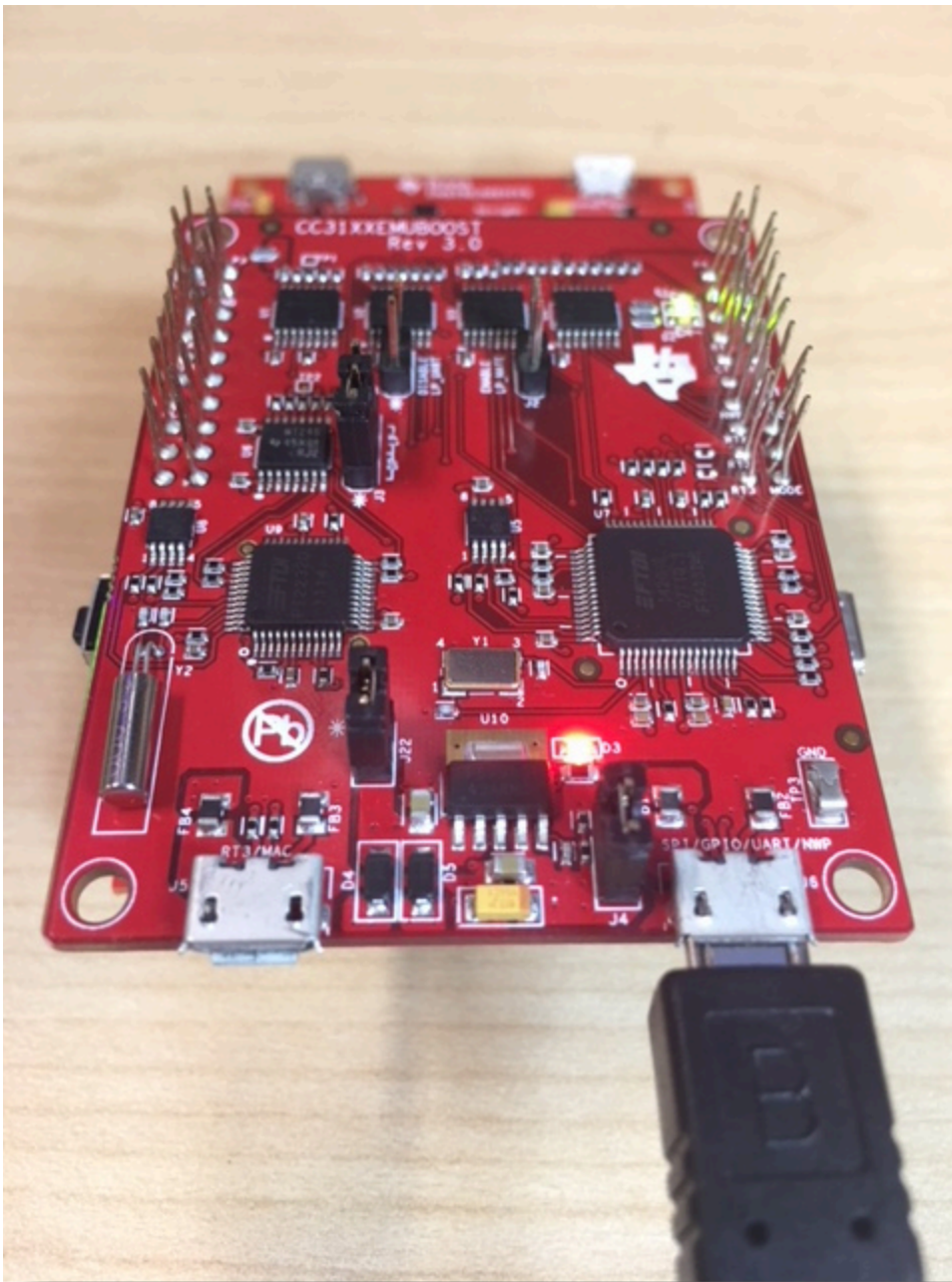
---

The most common and obvious way to update the user files on in CC31xx is to use the **C31XXEMUBOOT BoosterPack** along with **Uniflash's Image Creator**. The C31XXEMUBOOST is a specialized BoosterPack that connects on top of the CC31xx BoosterPack to program and update the user files stored on the CC31xx's flash memory. Uniflash has a specific utility (called Image Creator) that is used to communicate with the CC31XXEMU boost and create/manage software packages.

To get started first download the latest Uniflash package from the the link below:

[https://processors.wiki.ti.com/index.php/Category:CCS\\_UniFlash](https://processors.wiki.ti.com/index.php/Category:CCS_UniFlash)

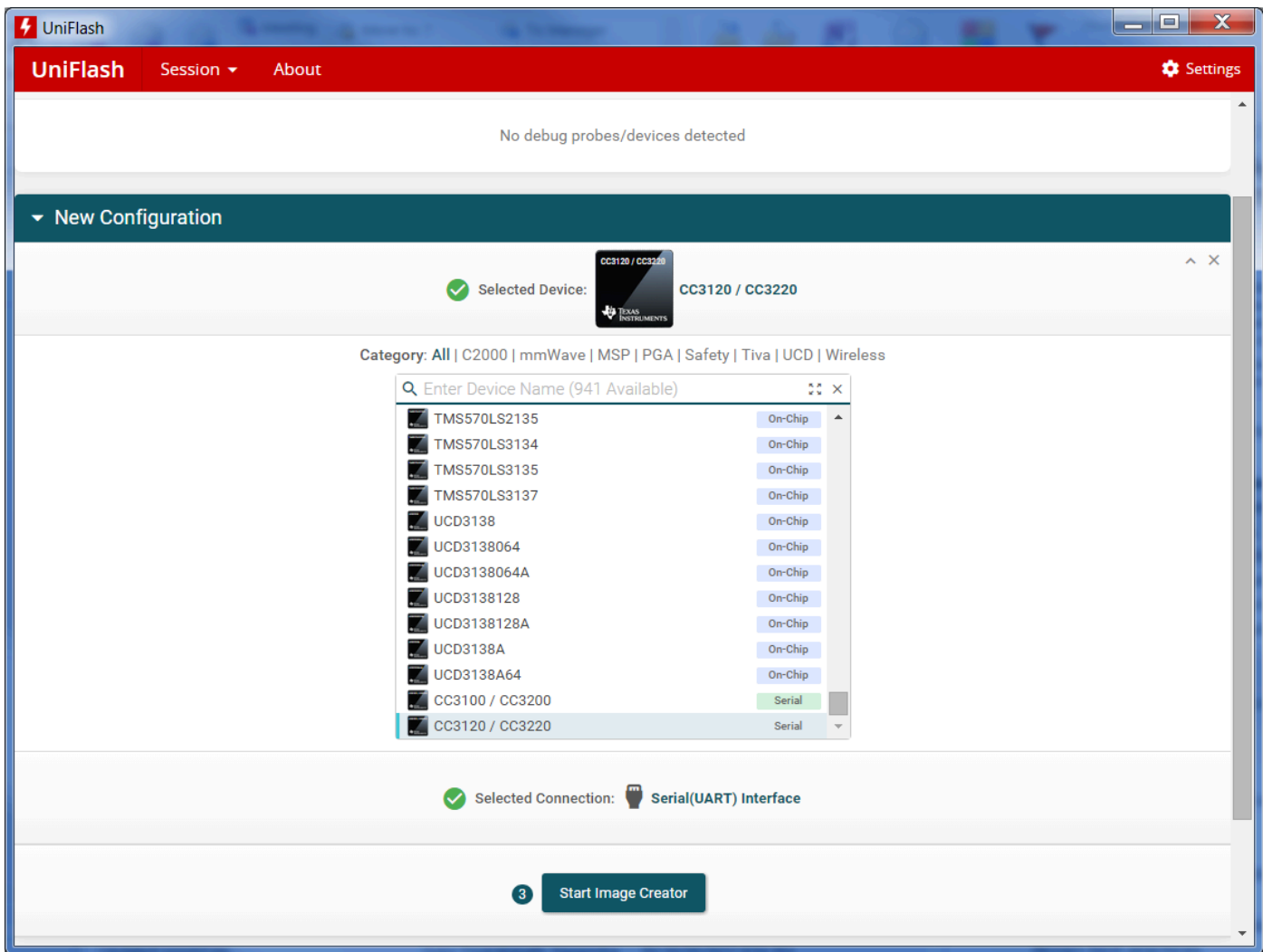
Next, connect the CC31XXEMUBOOST to the CC31xx BoosterPack as seen below while being careful not to mount the pins upside down.



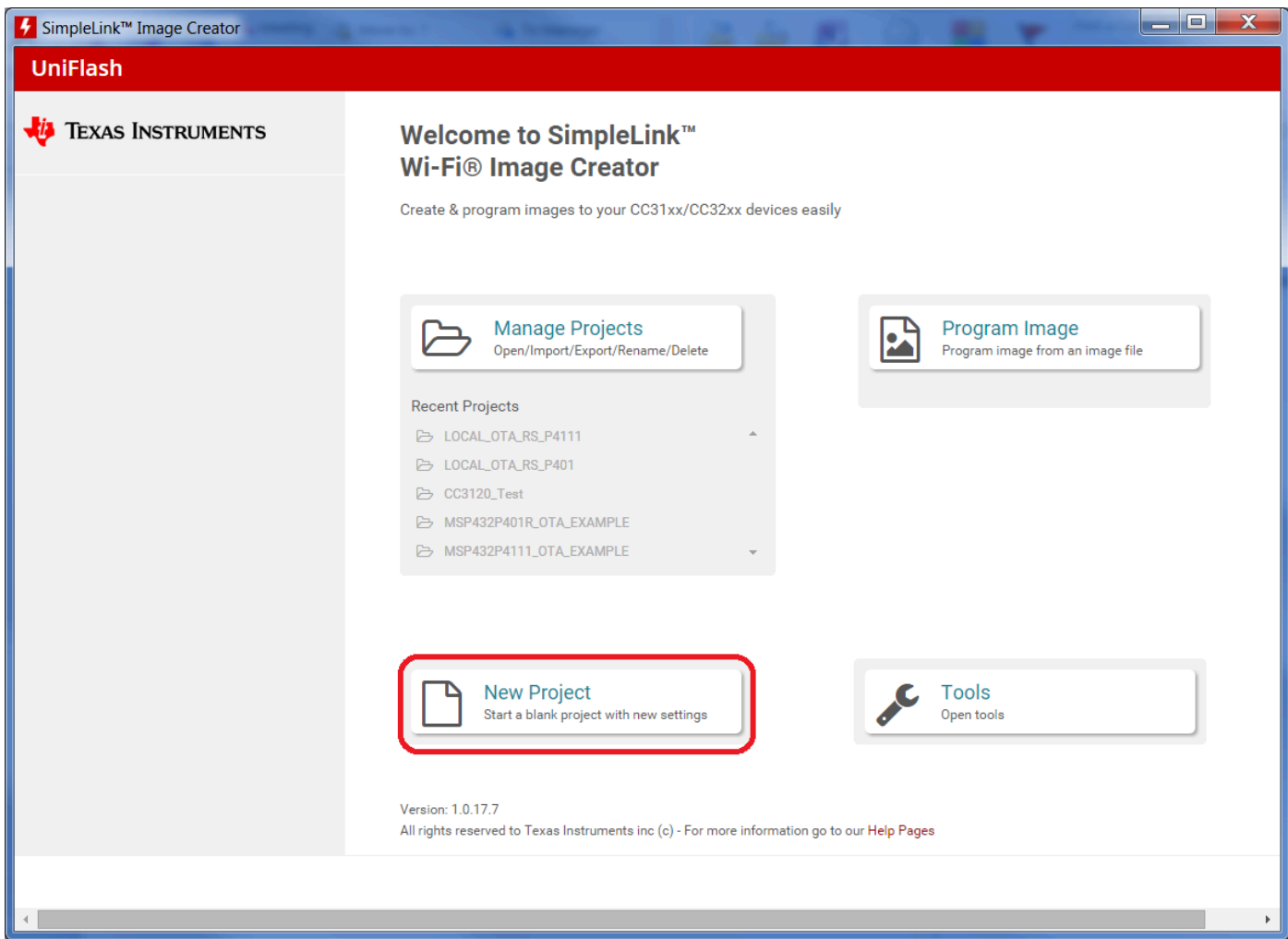
Note that while connected to the CC31XXEMUBOOST, the normal programming of the MSP432 LaunchPad will not work correctly. Also note that it is not required for the CC31xx to be connected to the MSP432 when programming with the CC31XXEMUBOOST.

Plug the USB cable from your computer into the right USB port of the CC31XXEMUBOOST (as seen above).

Once connected, open up the UniFlash application. In the device list, scroll down (to the very bottom) and select CC31xx/CC32xx.

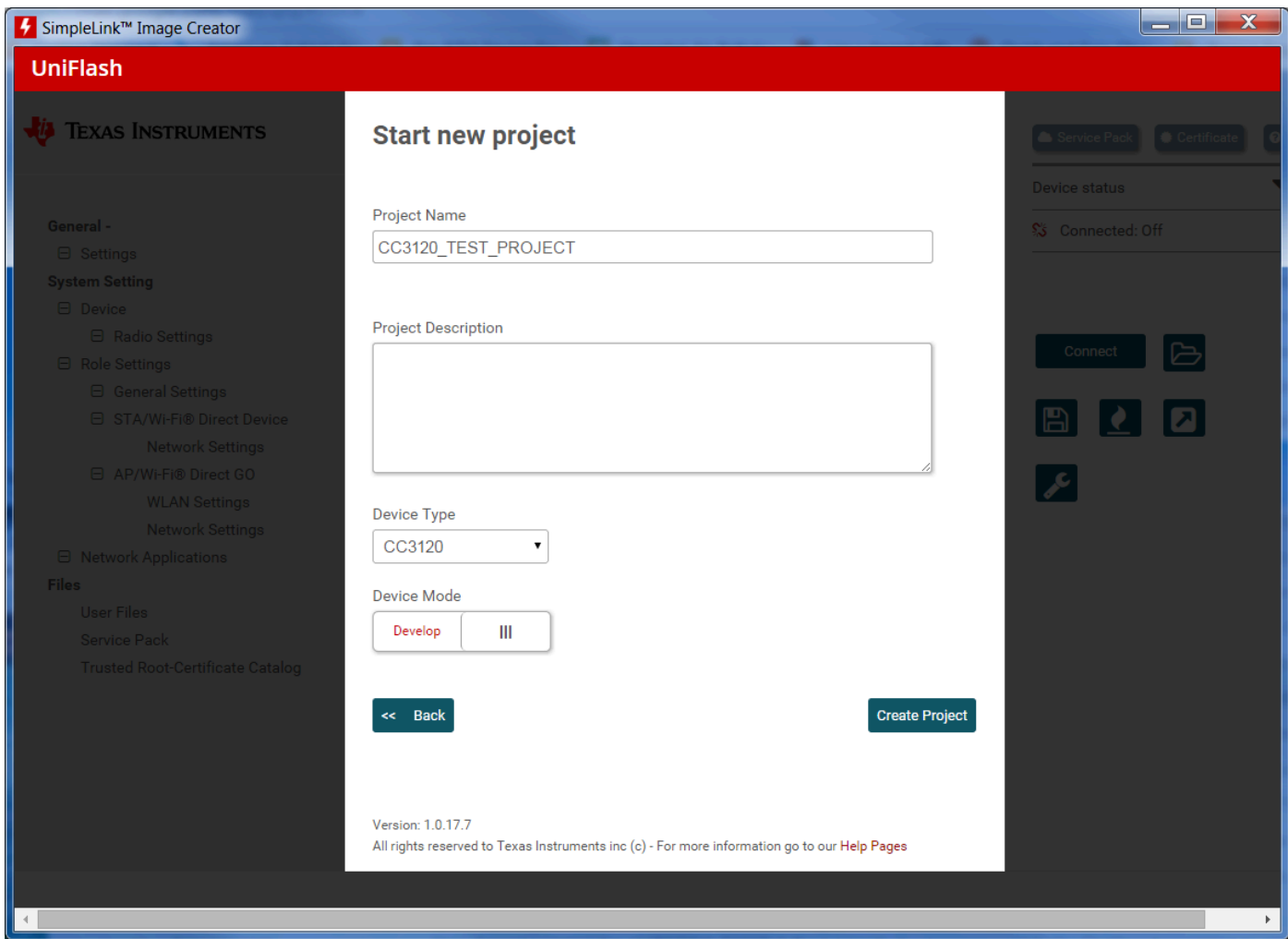


Once selected, click **Start Image Creator** to start the image creator application. This will start the SimpleLink Wi-Fi Image Creator. On this screen you can manage various project settings as well as program external images. To get started, click on the **New Project** button as seen below.

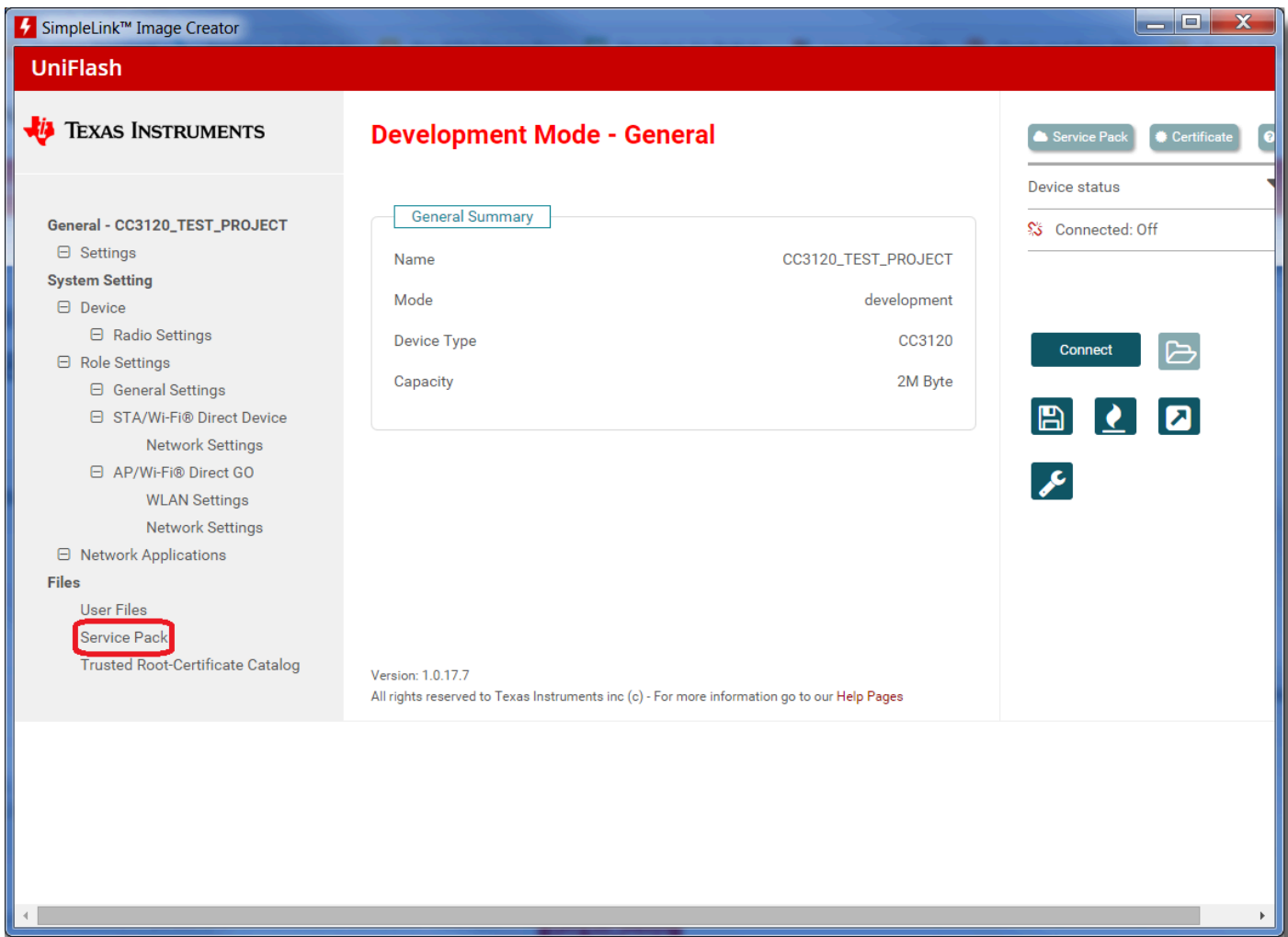


On the next screen fill out a project name of your choice. Make sure that you choose the BoosterPack that you are using (**CC3120** or **CC3135**) as the **Device Type** and make sure that the **Device Mode** is set to **Develop**. Develop mode allows the user to open up the filesystem and make changes via Uniflash during the development phase. When you wish to secure your device and program it for production the **Production** mode should be used.





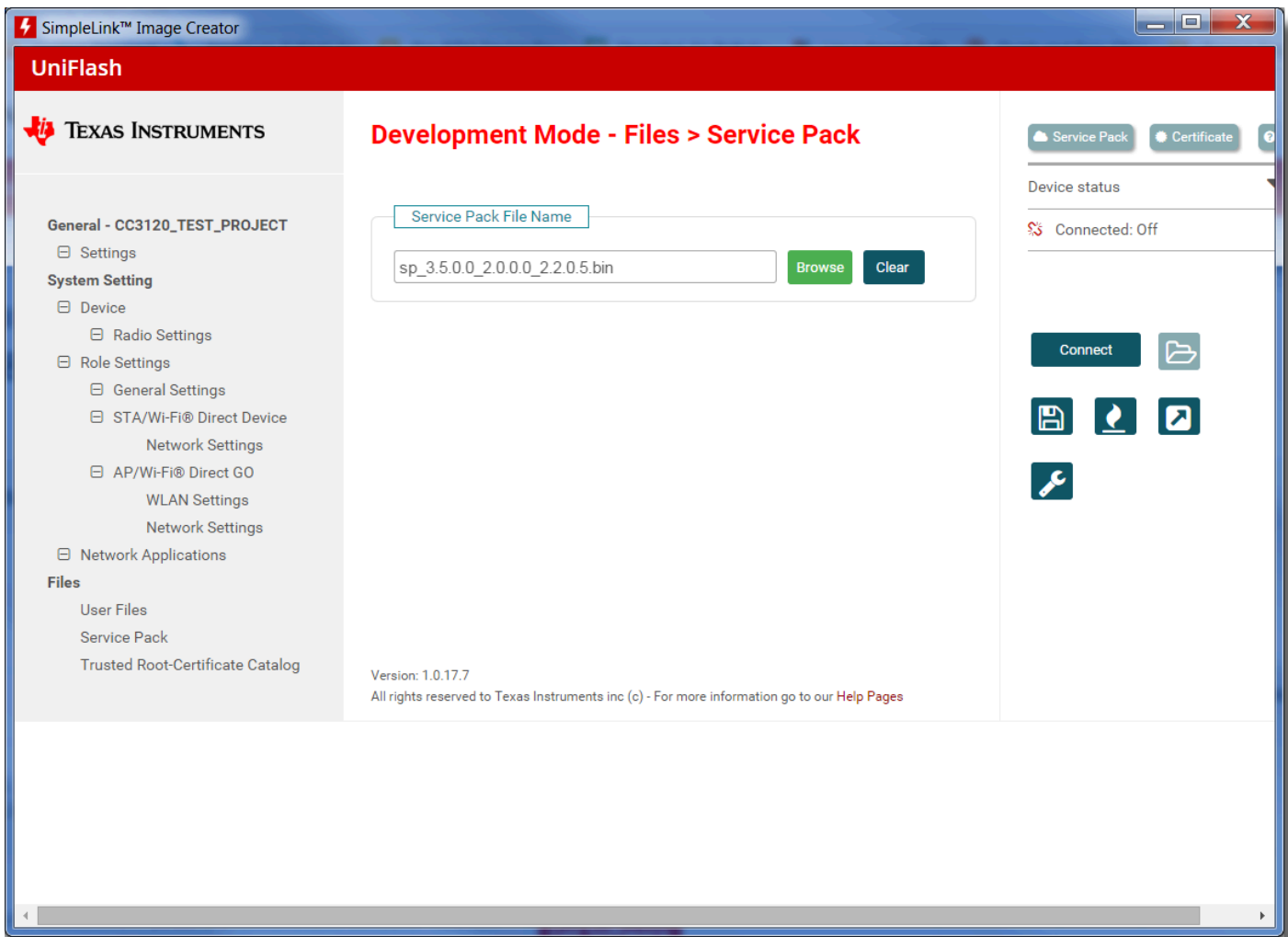
On the next screen you will see the new project page as it exists for Development Mode. In Development Mode you can change and manipulate all aspects of the CC31xx device. At a minimum when creating an image to flash on the CC31xx, it is recommended to start with specifying a service pack. The service pack provides various patches and security fixes for the NWP image running on the CC31xx. To specify a service pack select the **Service Pack** menu item on the left.



On the next page click browse next to the file name field and navigate to the service pack release that is included with the SimpleLink SDK Wi-Fi plugin release. The file name generally starts with **sp\_** and ends with **.bin**.

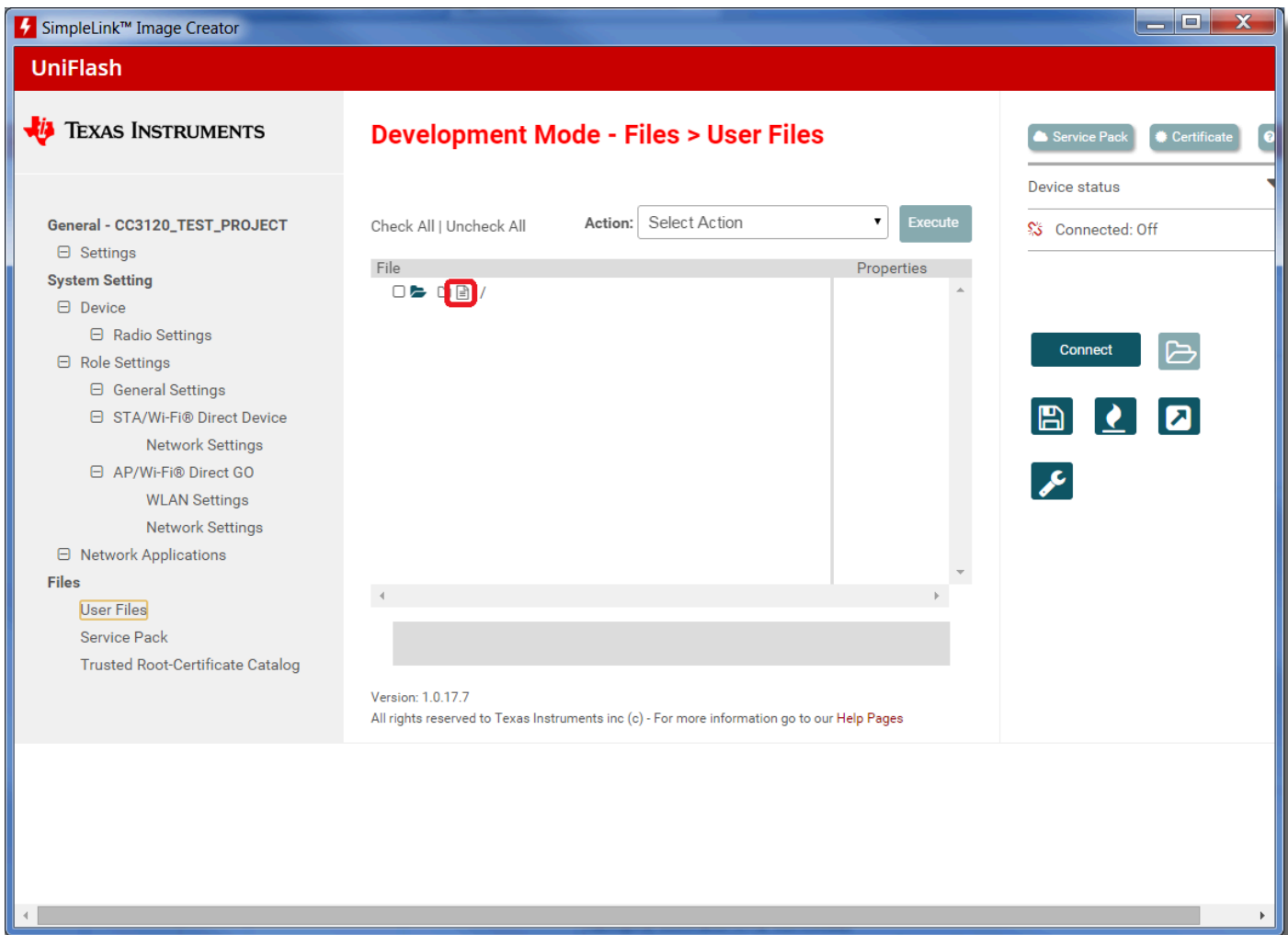
**Important Note:** Ensure that you are using the correct service pack for your device.

The service pack for the **CC3120** can be found at: `/tools/cc31xx_tools/servicepack-cc3x20`. The service pack for the CC3135 is located in the plugin in `tools/cc31xx_tools/servicepack-cc3x35`.

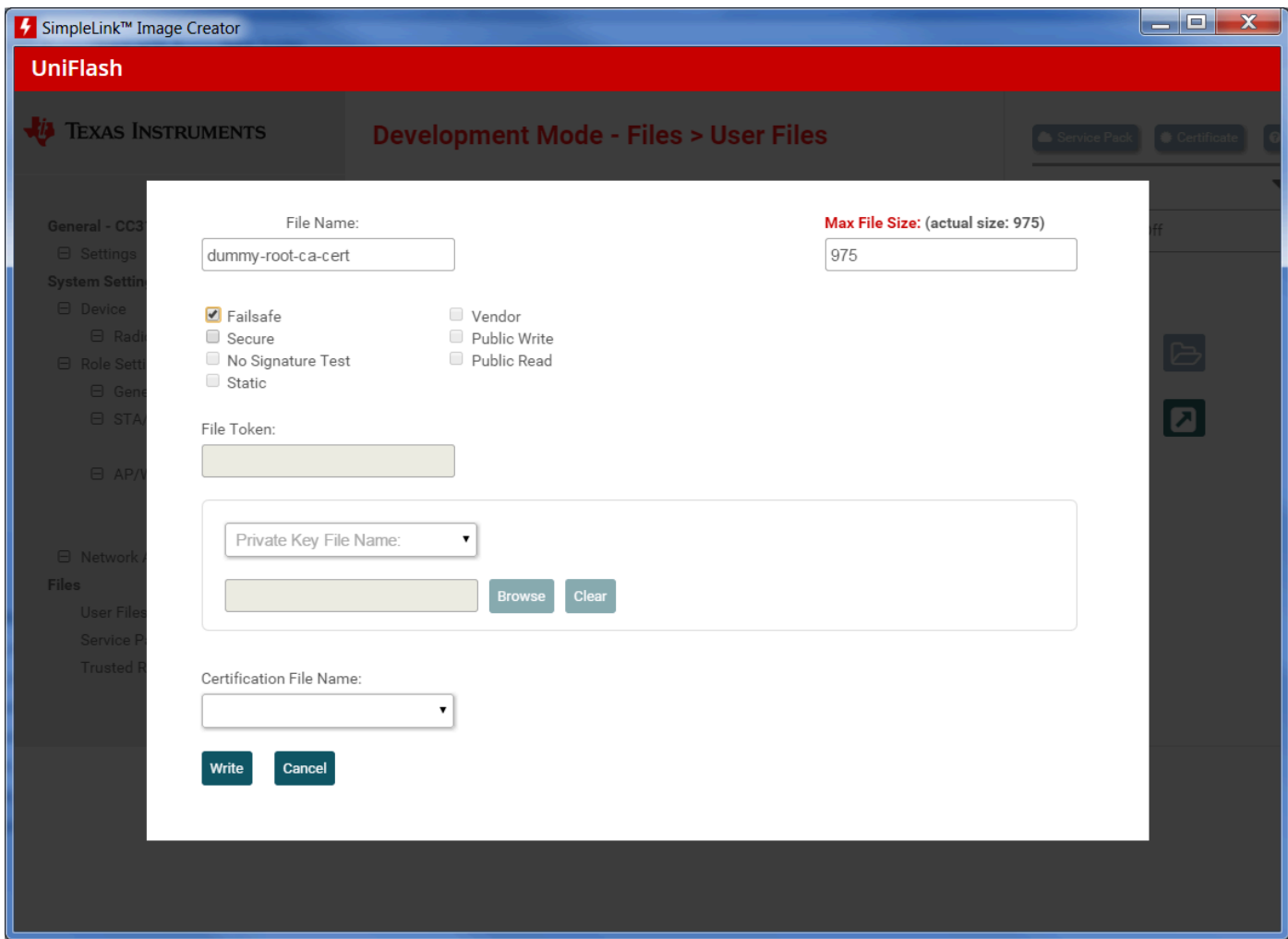


Note that when you select this file it persists within Uniflash- meaning if you save the project and reopen it the service pack file will exist within Uniflash's internal memory. The next portion of the project that is common to update is the user files section. The user files are all files that are user accessible that are stored on the flash memory of the CC31xx. These can include anything from application files such as html pages to certificates used for validation and security. To manage the user files contained within your image select the User Files menu item from the left.

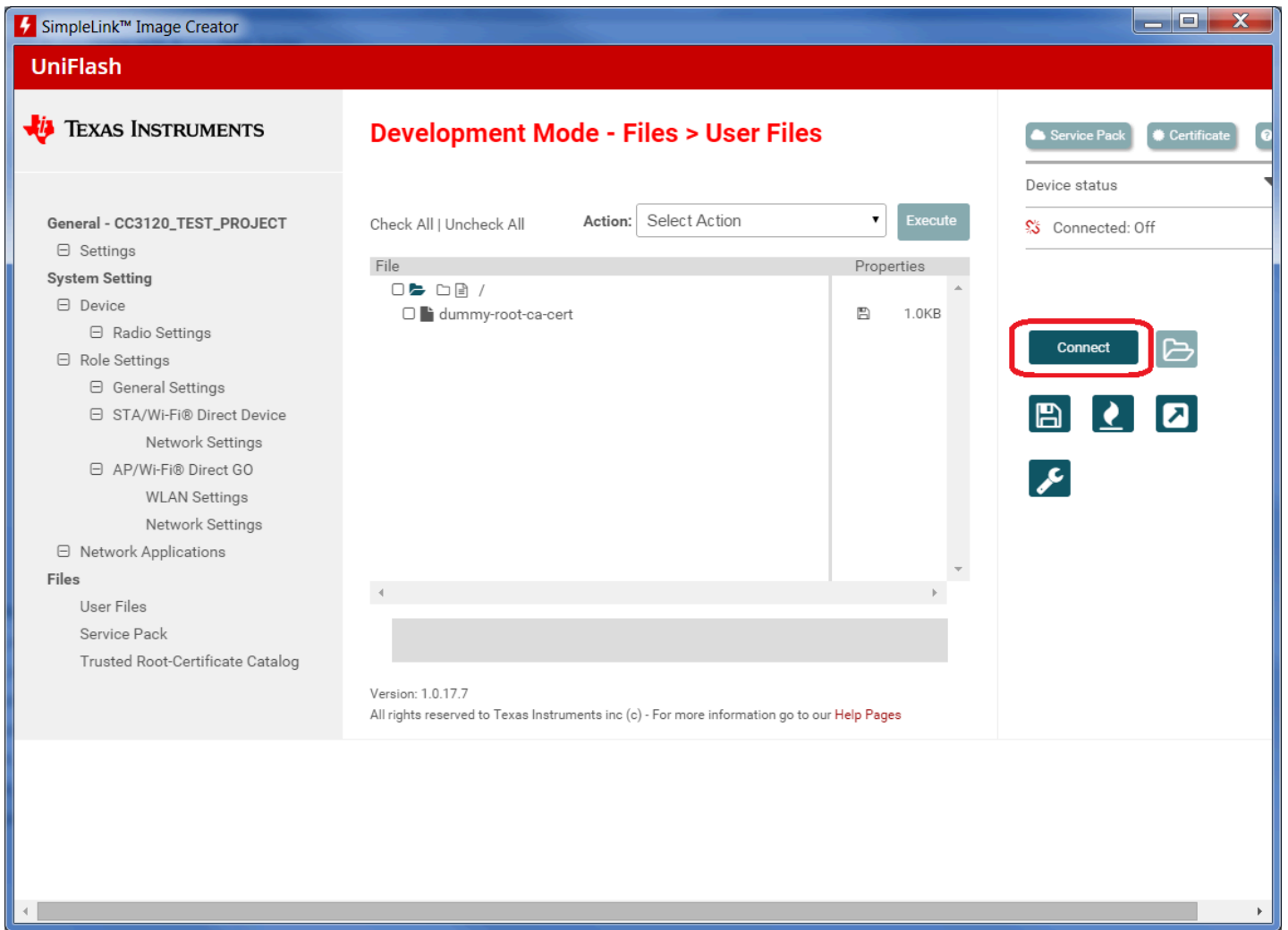
On the next screen the file manager view will be displayed. Since this is a new project there are no files currently populated. Let's add some files. Click on the small file icon located on the top of the file view.



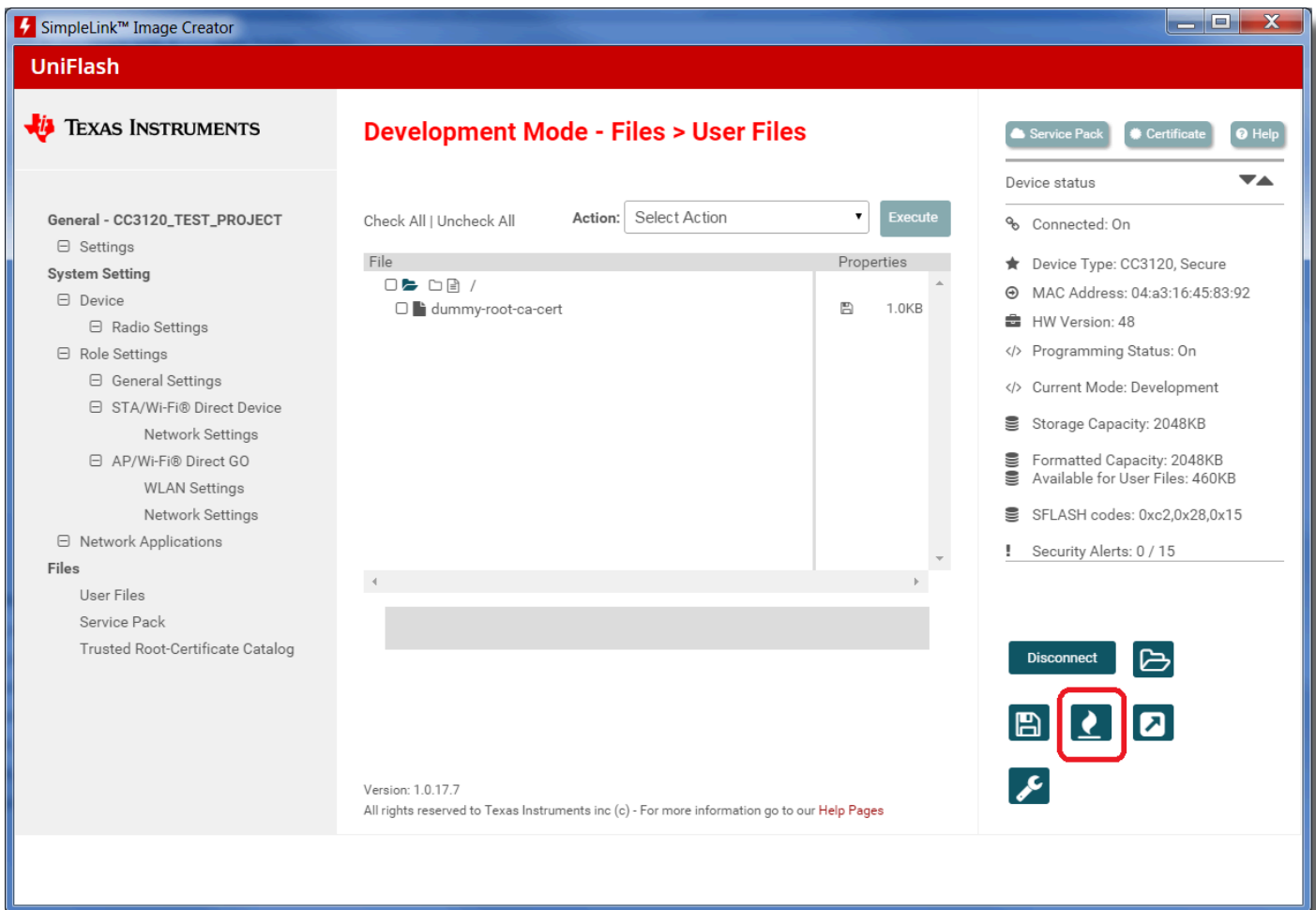
A common file to add to the user files is a root ca certificate. This certificate is responsible for securing a client connection and provides a level of security that ensures that users are legitimately connected to a trusted source. As part of the plugin release a “dummy” certificate is provided. To add this certificate to your project navigate to the `/tools/cc31xx_tools/certificate-playground` folder and select the **dummy-trusted-ca-cert** file to add to the project. On the file options window select **Failsafe** (this adds a certain level of redundancy for safety) and select **Write**. As stated before, the same user files can be used between the CC3120 and the CC3135.



As with the service pack, once a file is added to the Uniflash project it will persist in the Uniflash project (even if the file is deleted from the file system on your PC). Now it is time to program your CC31xx device. Note that when programming through Uniflash all files that are not included in the user files are erased and the new files are persisted. The exceptions are system files (such as the factory reset image). First connect to the device using the **Connect** button on the side.

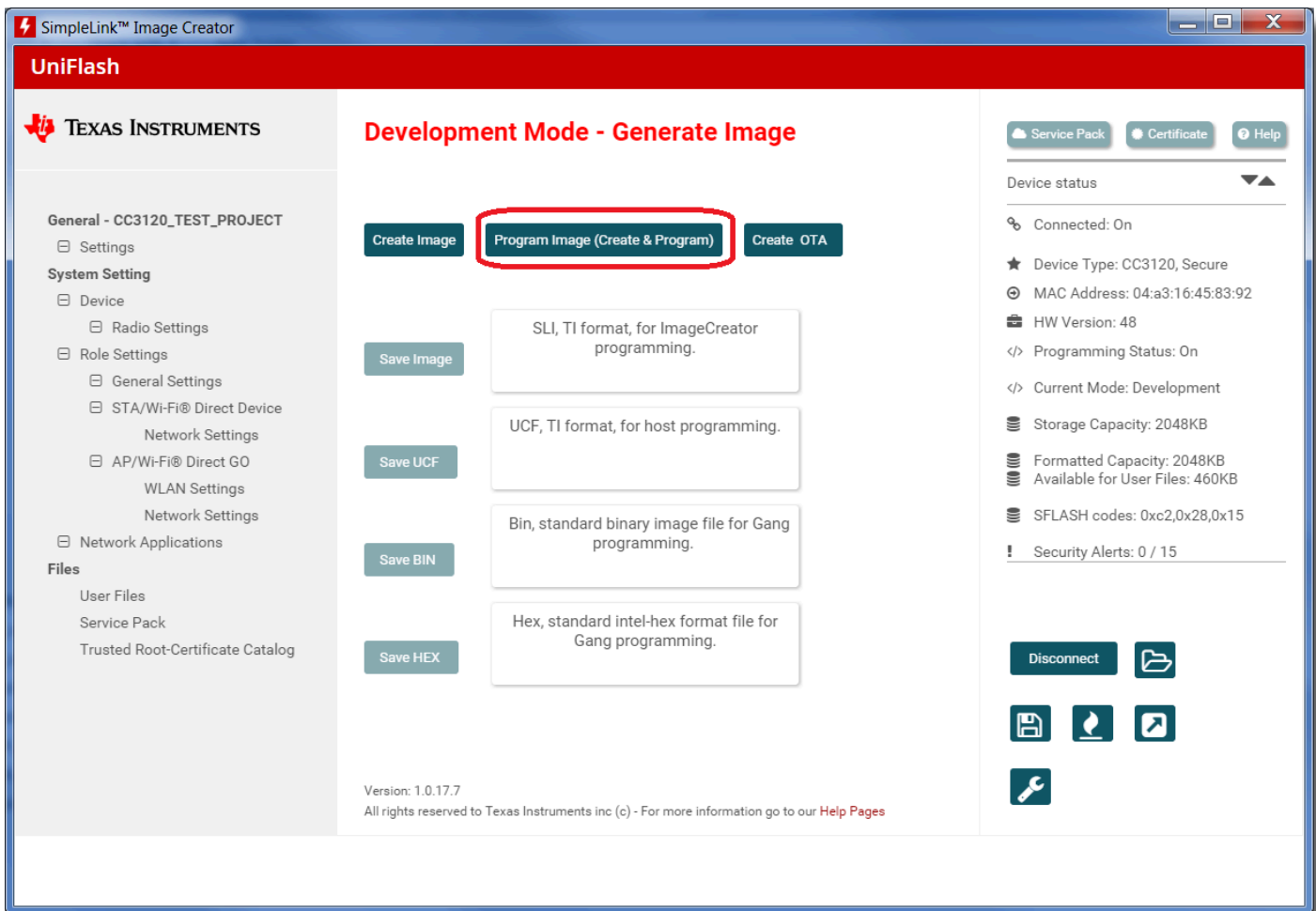


Once connected various information such as the MAC address and hardware version is displayed on the right. To program the image, first select the **Generate Image** icon on the right.



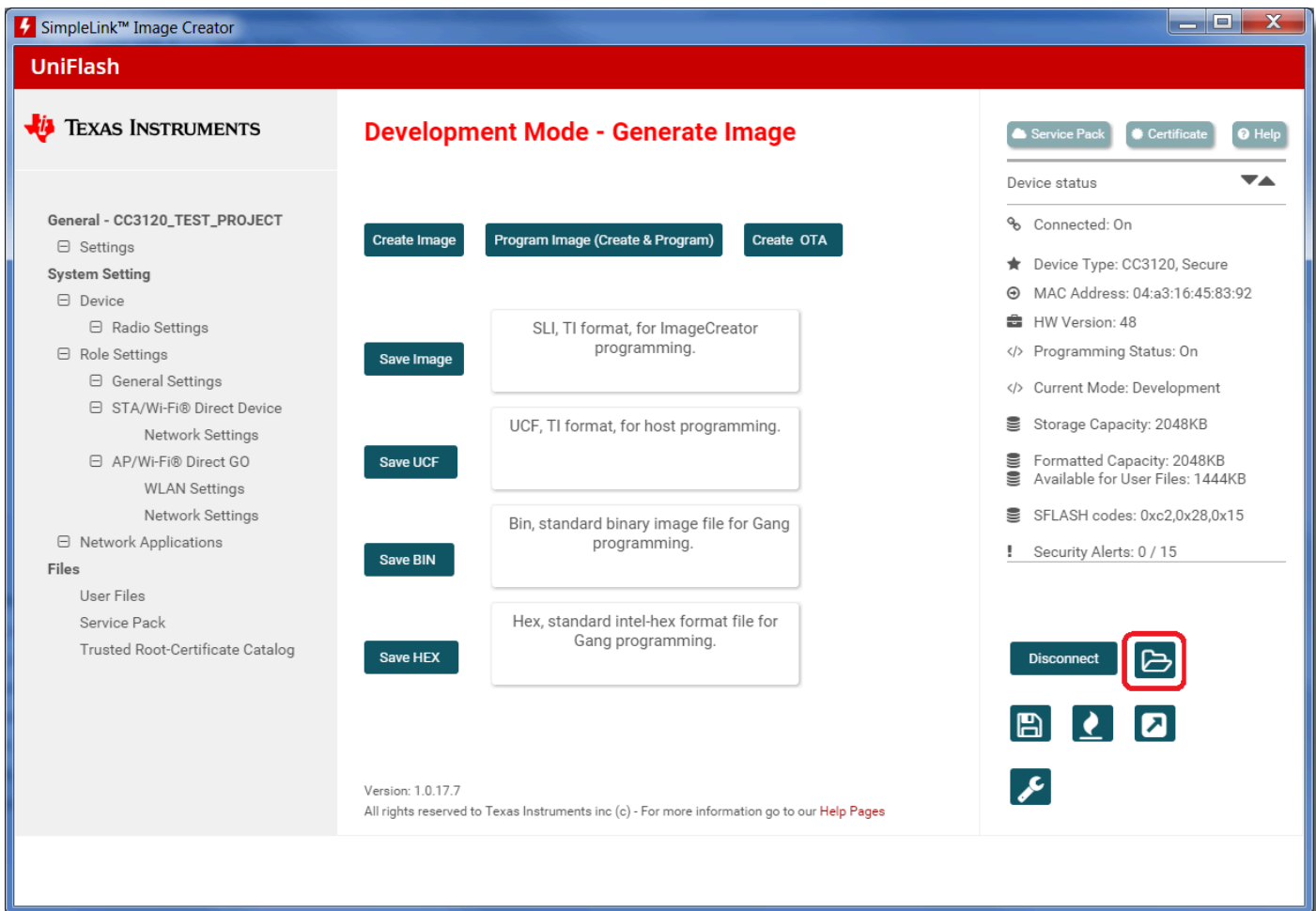
On the next screen select the **Program Image (Create & Program)**.

This will proceed to program the new image (both the service pack and the user file that you added) to the flash memory on the CC31xx BoosterPack.

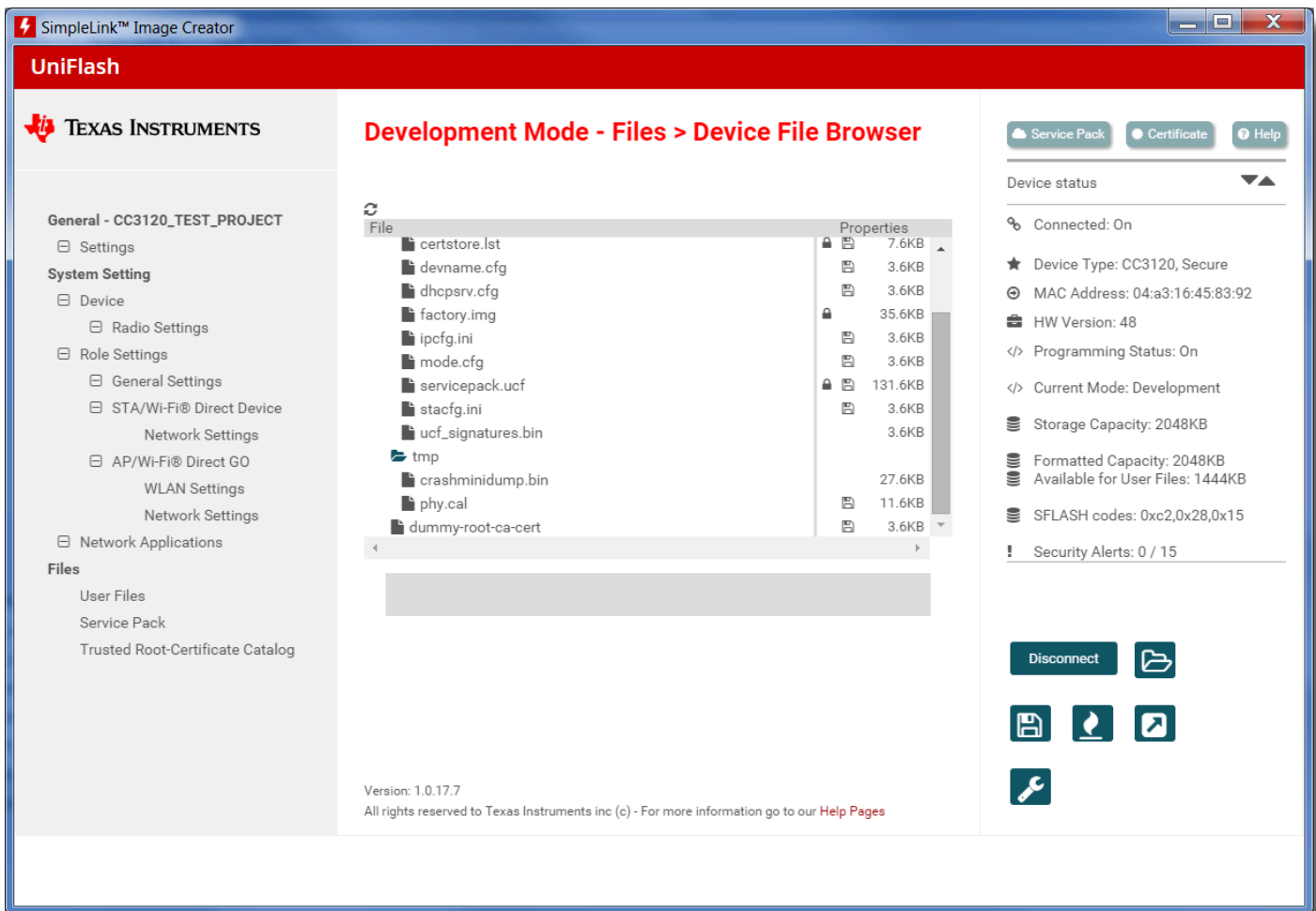


Once finished, click **Close** and then reconnect to the device using the **Connect** button from the previous step. After connected, select the **User File Browser** icon on the right next to the **Disconnect** button.





This will probe the CC31xx and display all files that currently exist on the flash file system. Other than the required system files, you will notice that the service pack file exists as `sys/servicepack.ucf` and that the `dummy-root-ca-cert` file exists in the root directory as programmed.



## Over-the-Air Updates

The second way to update user files on the CC31xx is to use an over-the-air (OTA) update scheme. OTA update is currently only supported on the MSP432P4 (MSP432P401R and MSP432P4111) family. This method uses the device's active Wi-Fi connection to dynamically download a prebuilt package and update relevant files on the network processor. This method also has the added benefit of being able to update the firmware image running on the MSP432 itself.

Two different code examples are provided that showcase the various methods of updating the firmware and user files over-the-air:

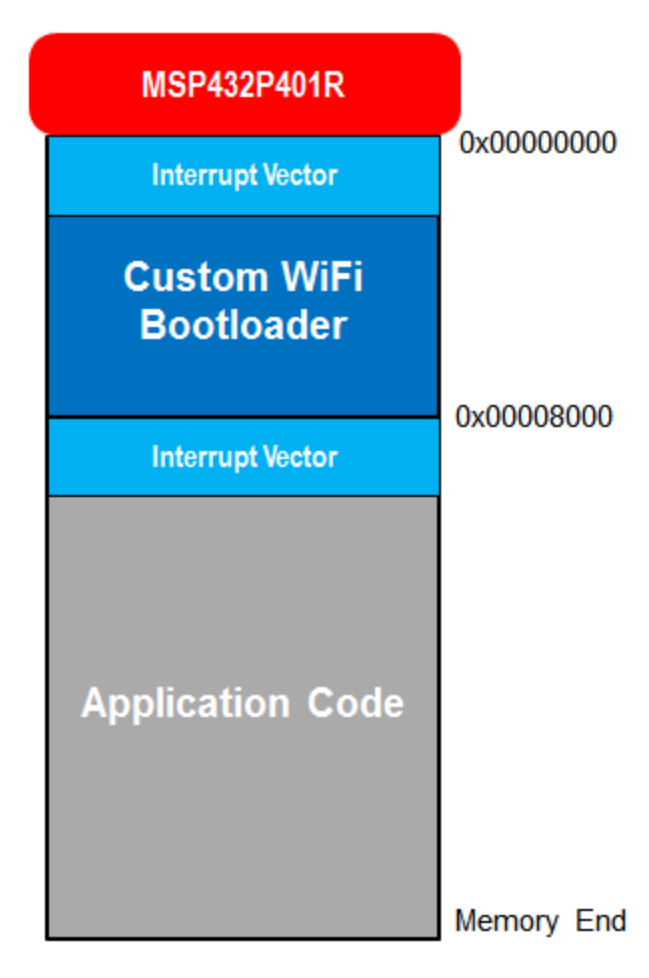
**Cloud OTA** – Update the firmware of the MSP432 and the service pack/user files of the CC31xx using cloud providers such as [GitHub](#) and [DropBox](#)

**Local OTA** – Update the firmware of the MSP432 and the service pack/user files of the CC31xx using a local HTTP server web interface

Both of these examples rely on having an image created in a specific format in order to operate correctly. These images are created using the [Uniflash](#) tool. It is important that the latest version of Uniflash is used (post 4.2.1) as the newer versions of Uniflash enable secure over-the-air firmware updates via SHA256 hash calculation.

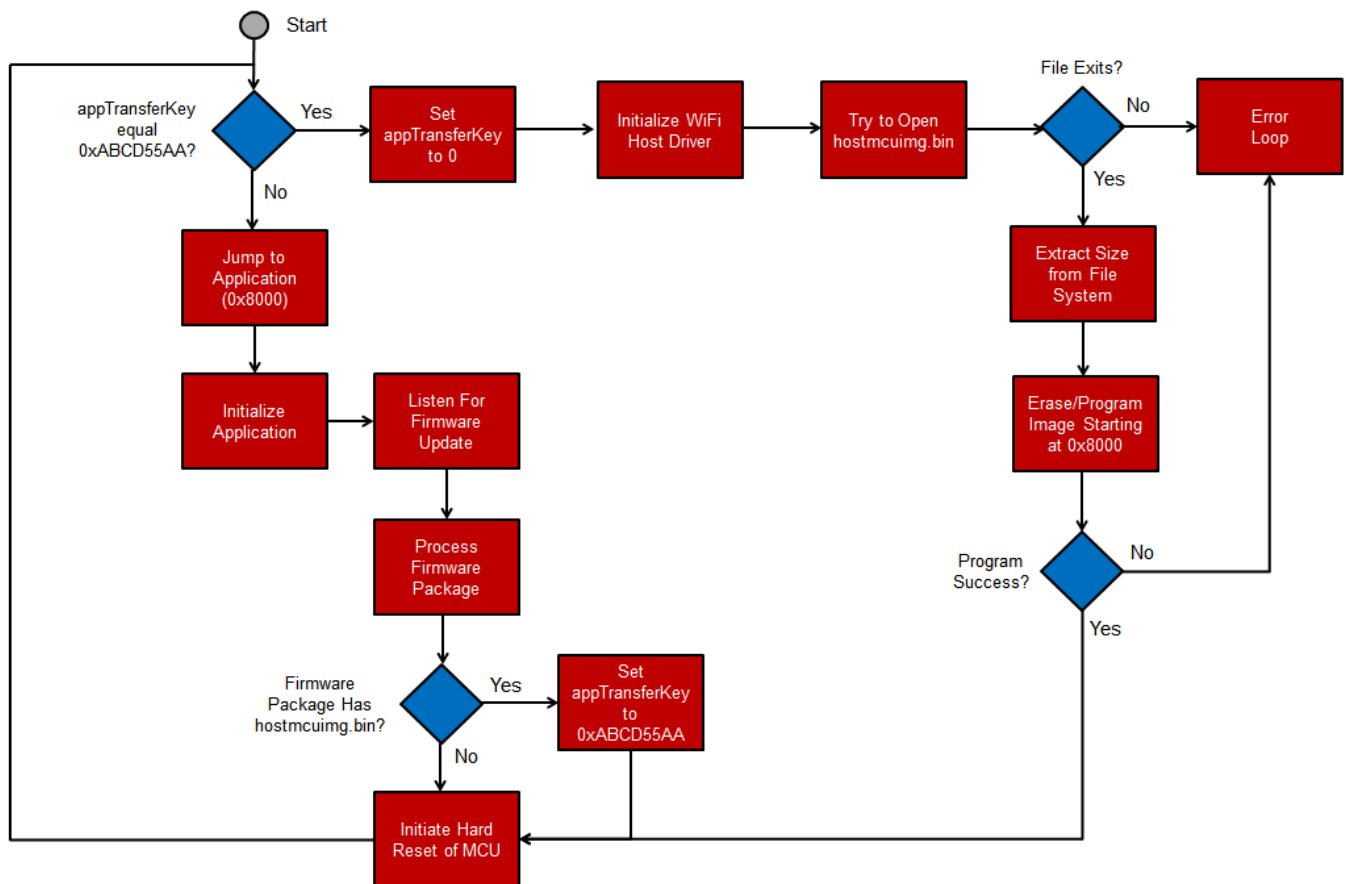
When creating a Uniflash OTA image the firmware image for the MSP432 must be placed at **/sys/hostmcuimg.bin** of the CC31xx's file system. Furthermore, the file must be marked with the **Failsafe** option in Uniflash during image creation. It is imperative to also have the OTA certificate file programmed in the root directory of the OTA image. This certificate is used for SHA256 validation and verification of the programmed image and the firmware update will fail in the absence of this certificate. A dummy certificate for demonstration purposes is located at **/tools/cc31xx\_tools/ota-example-cert/dummy\_ota\_vendor\_cert.der**

In an OTA enabled application for MSP432 (either MSP432P401 or MSP432P111) the first 32KB of flash memory is reserved for a customized bootloader. This bootloader will startup a small NoRTOS implementation of the CC31xx host driver, open the image programmed at **/sys/hostmcuimg.bin** on the CC31xx, and dynamically program that image to the MSP432's flash memory with a starting base address of **0x8000**. The memory map of a typical MSP432 OTA enabled device can be seen below. Note that this example is for an MSP432P401, however on an MSP432P4111 the structure will be the similar but adjusted for the MSP432P111's larger memory footprint.



Initially, when the Uniflash tar file is being programmed to the CC31xx over the active Wi-Fi connection, the firmware running on the MSP432 will detect if a new MSP432 firmware image exists in the package. If so, the MSP432's firmware will set a flag in a specific area of SRAM and initiate a hard reset of the device (causing execution to be handed over to the custom bootloader).

A flow chart of the custom OTA bootloader in the presence of a MSP432 firmware image can be seen below:



## Mobile Applications

Mobile applications are essential for provisioning an unconfigured SimpleLink device. “Provisioning” can be described as telling the SimpleLink device what router/access point to connect to and the relevant security information such as SSID/security key, etc. The various offerings of mobile applications provided by TI are described below.

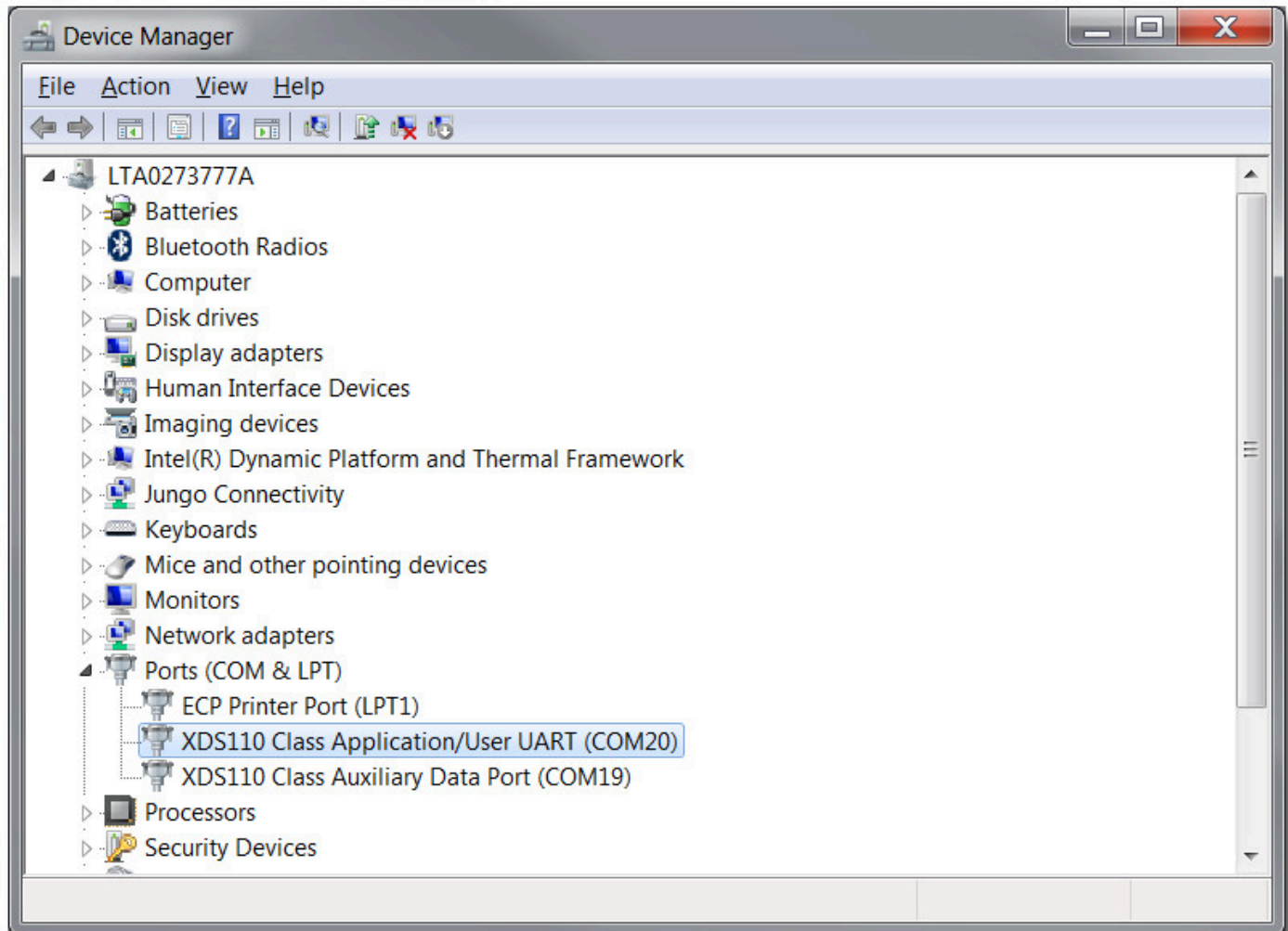
### SimpleLink SDK Starter Pro

SimpleLink SDK Starter Pro is an application written as way to specifically provision CC3xx family of devices (both Gen1 and Gen2). This applications supports both Android and iOS and supports advanced features such as SmartConfig and access point provisioning. Legacy versions of iOS are supported and the program (for iOS) is written entirely in ObjectiveC. To download and use SimpleLink SDK Starter Pro please refer to the link below:

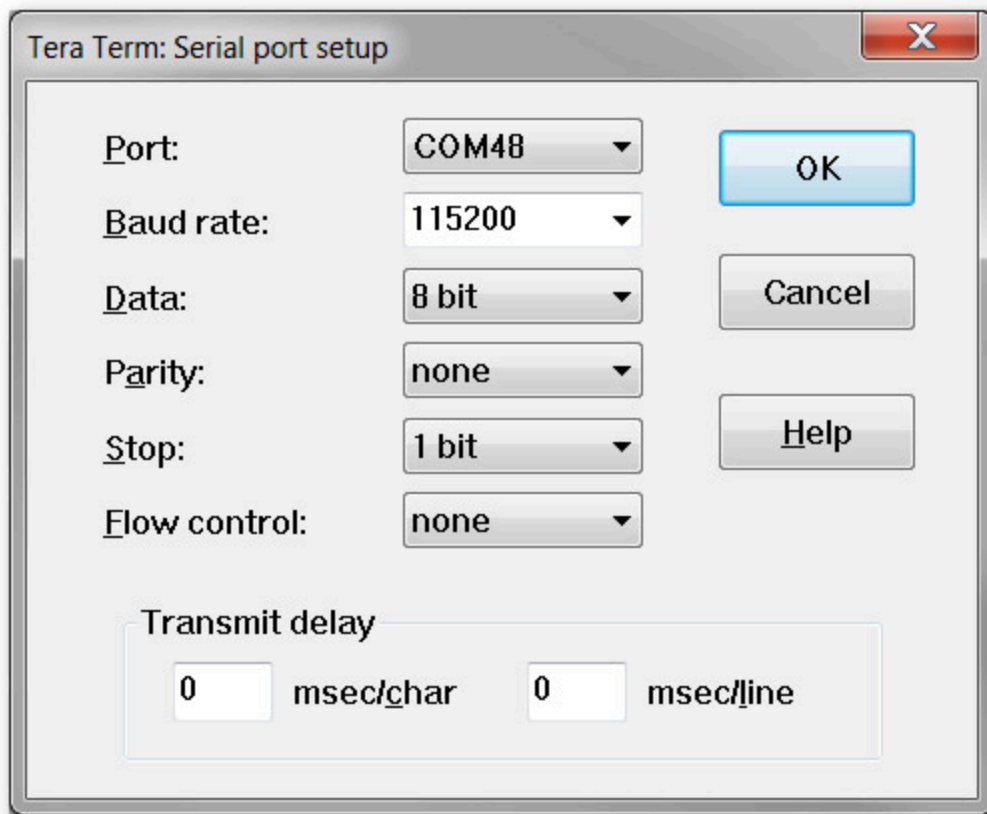
<https://www.ti.com/tool/wifistarterpro>

## Serial Port Logging/Debugging

Note that for logging/debugging messages the back channel serial port of the LaunchPad is used. In order to read these messages, use your favorite serial terminal program (such as [Tera Term](#) or [PuTTY](#)). The COM port number will be different depending on the system configuration, however it should show up as *XDS110 Class Application/User UART*.



Make sure to use the Application/User UART and not the Auxiliary Data Port. Along with the COM port of the XDS110 UART, the back channel UART defaults to the following serial port values:



## Customizing Parameters

As the SimpleLink SDK Wi-Fi Plugin leverages a two chip solution that uses SPI as a communications vessel between the two chips, it is necessary to have an easily accessible way to configure the SPI parameters from the host MCU's firmware. For this reason an additional configuration parameter was added to the board file of each application example that controls the various configuration settings for the master SPI interface.

```
/*
 * ===== Wi-Fi =====
 *
 * This is the configuration structure for the Wi-Fi module that will be used
 * as part of the SimpleLink SDK Wi-Fi plugin. These are configured for SPI mode.
 */
#include <ti/drivers/net/wifi/porting/SIMPLELINKWIFI.h>

const SIMPLELINKWIFI_HWAttrsV1 wifiSimplelinkHWAttrs =
{
    .spiIndex = MSP_EXP432P4111_SPIB0,
    .hostIRQPin = MSP_EXP432P4111_HOST_IRQ,
    .nHIBPin = MSP_EXP432P4111_nHIB_pin,
    .csPin = MSP_EXP432P4111_CS_pin,
    .maxDMASize = 1024,
    .spiBitRate = 3000000
};

const uint_least8_t WiFi_count = 1;
```

```
const WiFi_Config WiFi_config[1] =
{
{
    .hwAttrs = &wifiSimplelinkHWAttrs,
}
};
```

This example is shown for an MSP432P4111, however the parameters will be similar for both the other SimpleLink devices. Here the SPI bit rate, the physical pins used for the various CC31xx signals, and the buffer size for DMA transfers between the host and CC31xx can be specified. Note that on MSP432 devices that 1024 is the maximum size that can be used for DMA transfers.

## Code Examples

**Cloud OTA** – Update the firmware of the MSP432 and the service pack/user files of the CC31xx using cloud providers such as [GitHub](#) and [DropBox](#)

**Local OTA** – Update the firmware of the MSP432 and the service pack/user files of the CC31xx using a local HTTP server web interface

**MQTT Client** – Example implementation of an [MQTT](#) client communicating to a cloud MQTT cloud server

**MQTT Client Server** – MQTT server implementation that demonstrates use of CC31xx in both MQTT server and client roles

**Network Terminal** – Full featured code example that allows users to configure and manipulate all aspects of the CC31xx through an interactive UART terminal

**Power Measurement** – Showcases power numbers for various modes of operation and hibernation modes.

**Provisioning** – Interacts with various host provisioning applications to provision the CC31xx to a central access point/router

**Trigger Mode** – Demonstrates how to use “trigger mode” to awake the device from hibernation using a specific packet on a predefined socket

**TCP Echo using Ethernet and Wi-Fi** – Showcases the tandem use of the Ethernet module and the Wi-Fi module using a TCP echo.

**Wi-Fi Ethernet Sockets** – Showcases how SNetSock API's will choose the correct interface on an automatic setting.

**MSP432P401 BootLoader** – Source code of the custom MSP432P401 bootloader used to update the firmware of the MSP432 over Wi-Fi

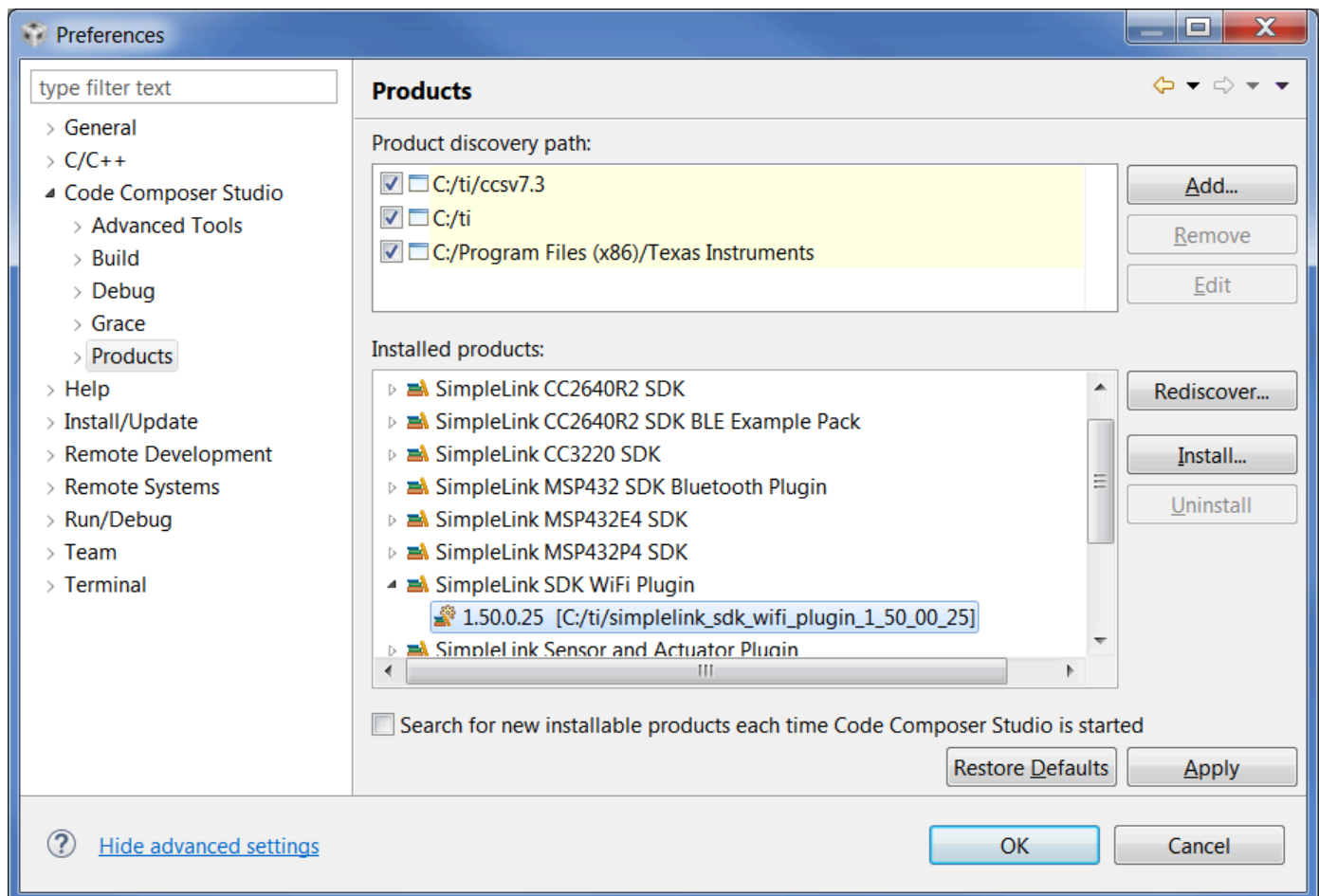
**MSP432P4111 BootLoader** – Source code of the custom MSP432P4111 bootloader used to update the firmware of the MSP432 over Wi-Fi

# IDE/Software Configurations

This software package supports and has been tested with TI Code Composer Studio 8.2 as well as IAR Embedded Workbench for ARM v8.20.2. The GCC toolchain for ARM has also been tested within CCS as well as directly through command line invocation. Both FreeRTOS and TI-RTOS are fully supported in this release as well as addition “NoRTOS” support.

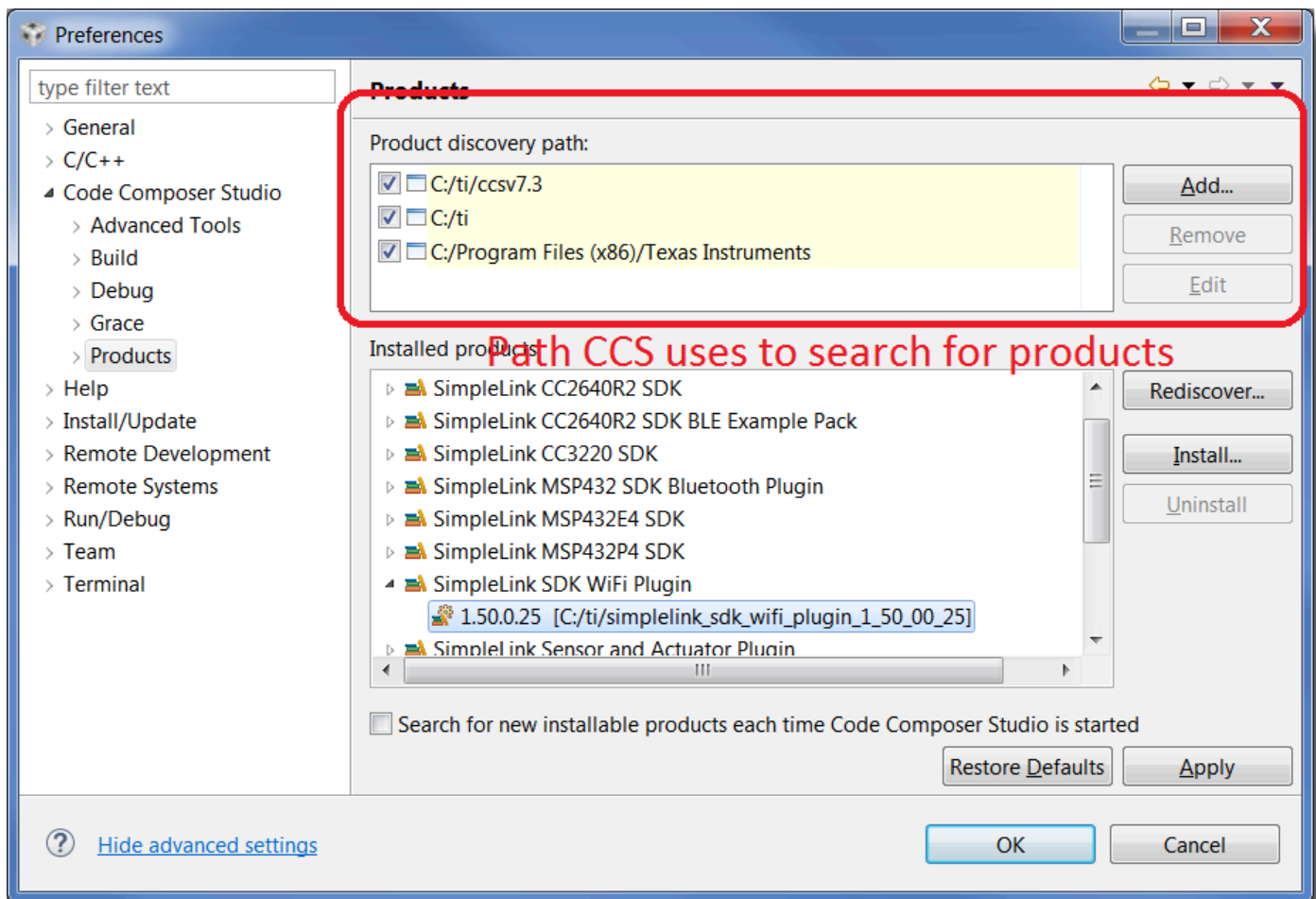
## Code Composer Studio Project Guide

The SimpleLink SDK Wi-Fi Plugin is designed to integrate seamlessly with the [Code Composer Studio Integrated Development Environment](#) from Texas Instruments. After installing the plugin (it is recommended to use the default installation directory), CCS should automatically detect and install the plugin software package without any special interaction from the user. To check to see if the plugin has installed correctly, go to the **Windows->Preferences** menu item. Under the **Code Composer Studio->Products** item you should see “SimpleLink SDK Wi-Fi Plugin” installed (with a version number):



Note that the installation path of the plugin needs to be present in the Product Discovery Path:

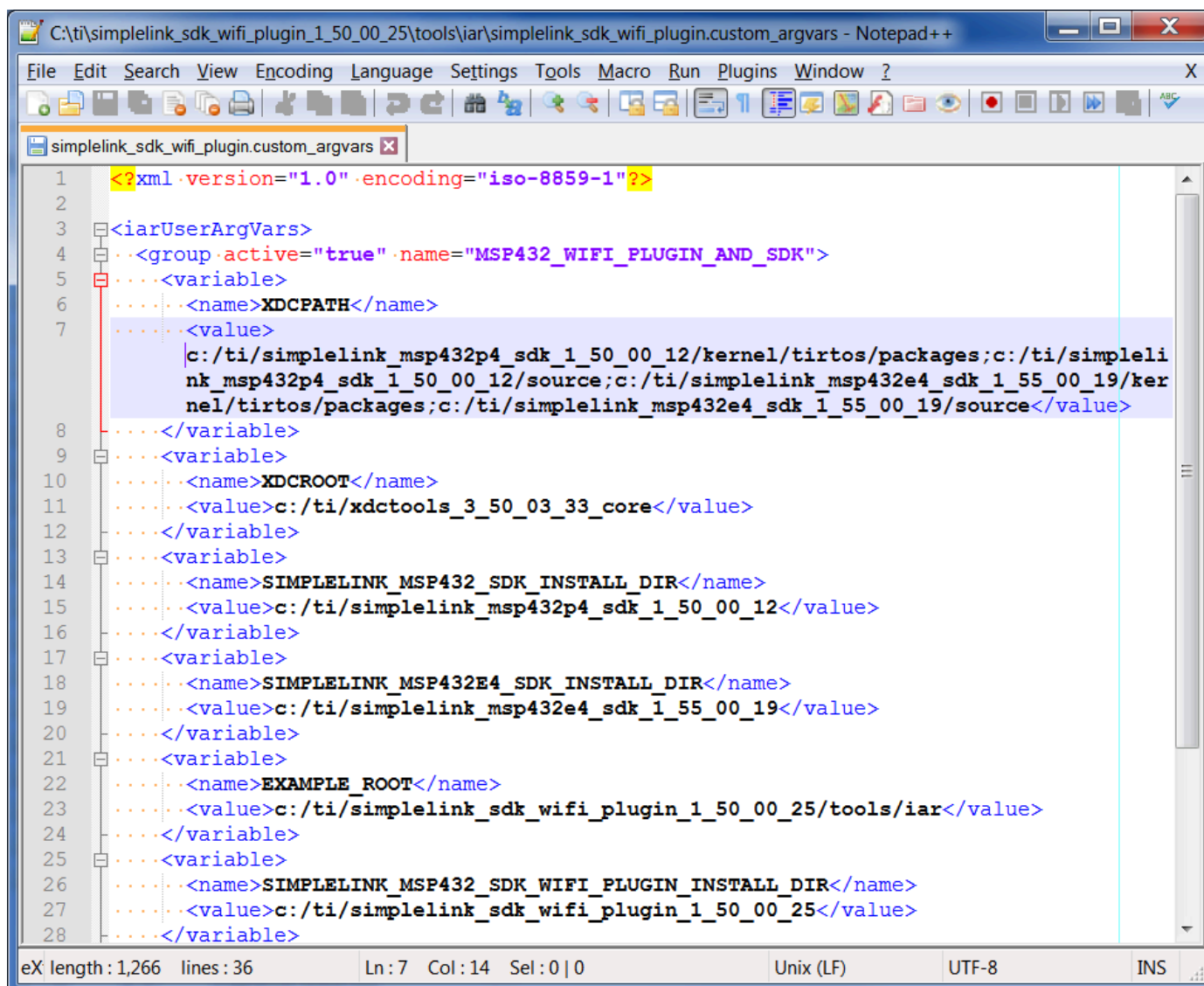




## IAR Embedded Workbench Project Guide

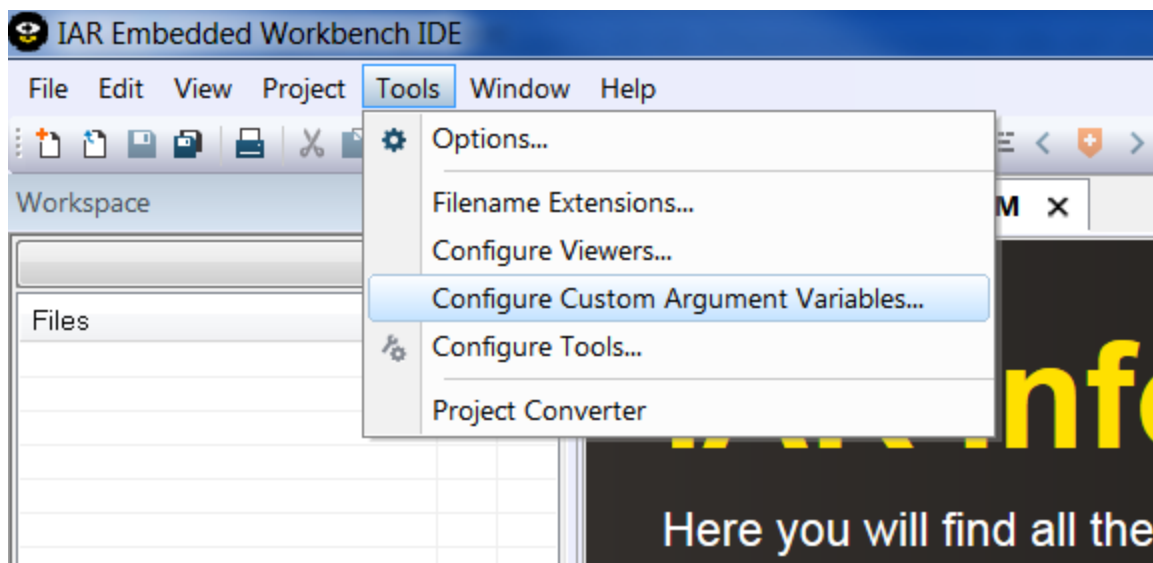
IAR Embedded Workbench is fully supported under the SimpleLink SDK Wi-Fi Plugin ecosystem. Setting up your development environment to work with IAR involves pointing IAR to the plugin and SDK directories as mentioned below.

To get started, a list of external variables needs to be imported in IAR in order to let the IDE know the paths for all of the relevant software installations. An example configuration file that uses the default paths can be found in the `tools/iar/simplelink_sdk_wifi_plugin.custom_argvars` file of the SimpleLink SDK Wi-Fi Plugin installation directory. This file might have to be changed depending on specific installation paths and version numbers; however it should work if the user chose to use the default installation paths (note the version numbers might slightly vary from the picture below):

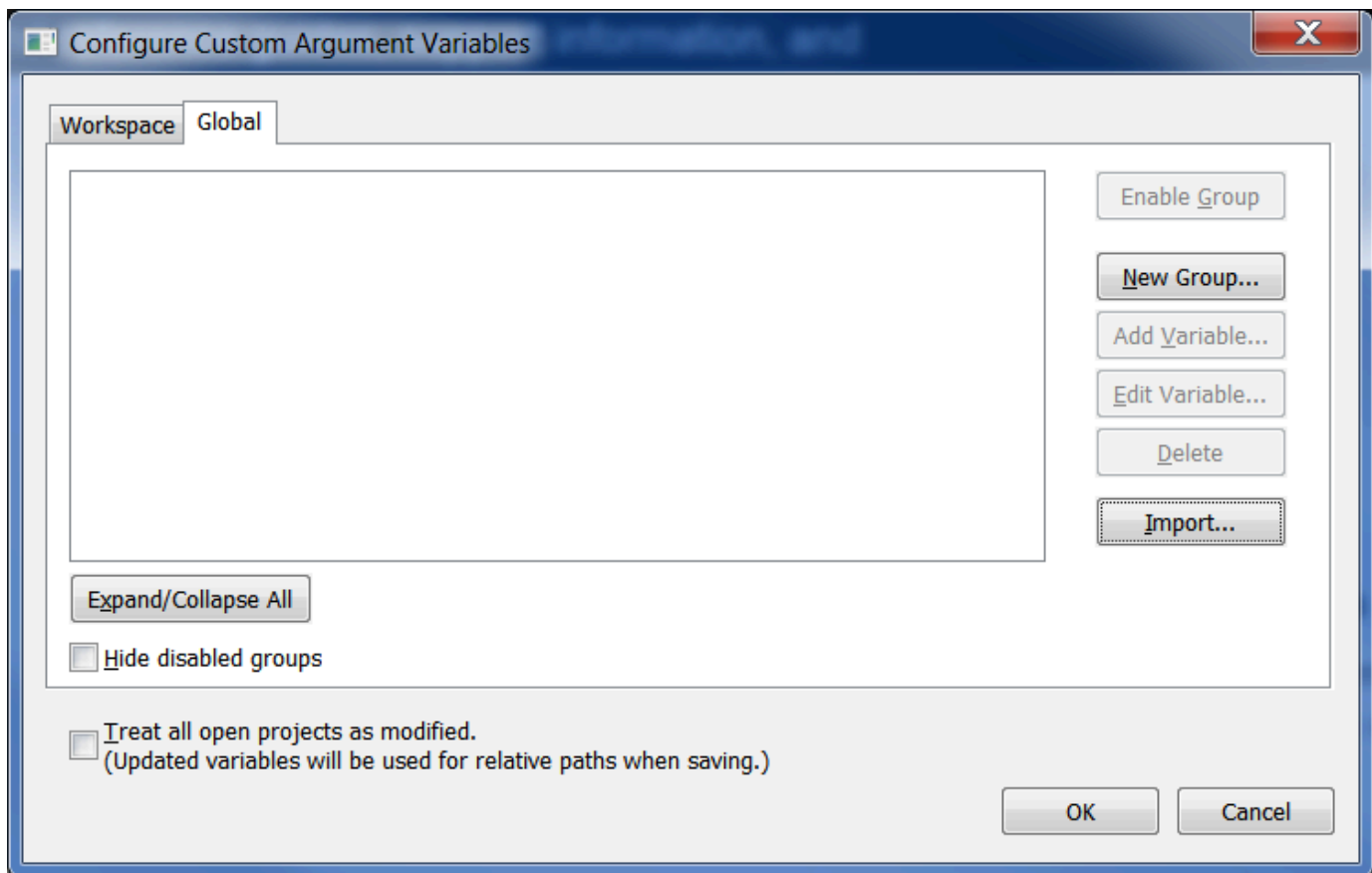


```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2
3 <iarUserArgVars>
4   <group active="true" name="MSP432_WIFI_PLUGIN_AND_SDK">
5     <variable>
6       <name>XDCPATH</name>
7       <value>
8         c:/ti/simplelink_msp432p4_sdk_1_50_00_12/kernel/tirtos/packages;c:/ti/simpleli
9         nk_msp432p4_sdk_1_50_00_12/source;c:/ti/simplelink_msp432e4_sdk_1_55_00_19/ker
10        nel/tirtos/packages;c:/ti/simplelink_msp432e4_sdk_1_55_00_19/source</value>
11     </variable>
12     <variable>
13       <name>XDCROOT</name>
14       <value>c:/ti/xdctools_3_50_03_33_core</value>
15     </variable>
16     <variable>
17       <name>SIMPLELINK_MSP432_SDK_INSTALL_DIR</name>
18       <value>c:/ti/simplelink_msp432p4_sdk_1_50_00_12</value>
19     </variable>
20     <variable>
21       <name>SIMPLELINK_MSP432E4_SDK_INSTALL_DIR</name>
22       <value>c:/ti/simplelink_msp432e4_sdk_1_55_00_19</value>
23     </variable>
24     <variable>
25       <name>EXAMPLE_ROOT</name>
26       <value>c:/ti/simplelink_sdk_wifi_plugin_1_50_00_25/tools/iar</value>
27     </variable>
28     <variable>
29       <name>SIMPLELINK_MSP432_SDK_WIFI_PLUGIN_INSTALL_DIR</name>
30       <value>c:/ti/simplelink_sdk_wifi_plugin_1_50_00_25</value>
31     </variable>
32   </group>
33 </iarUserArgVars>
```

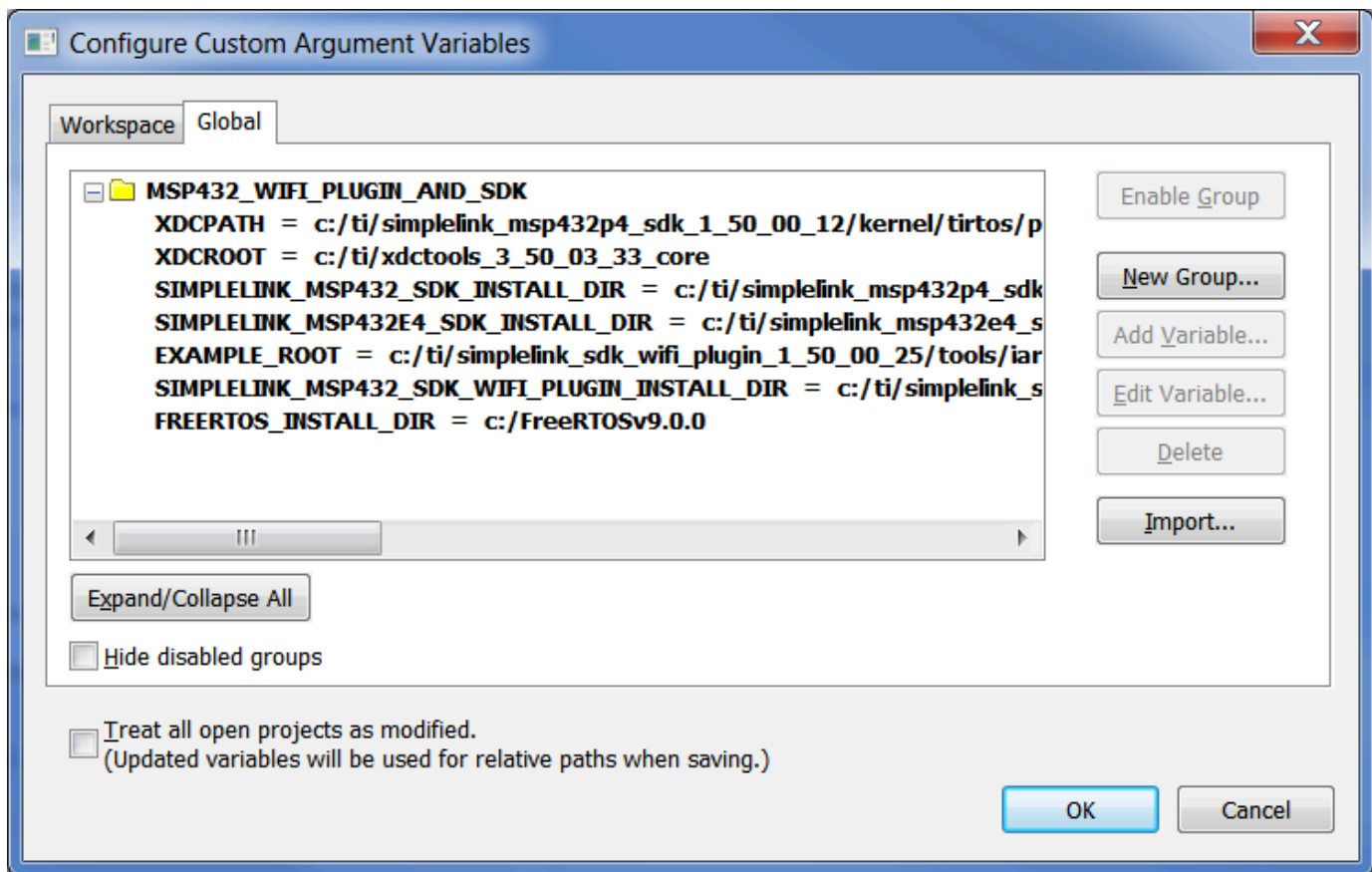
To import this file into your IAR IDE, navigate to *Tools->Configure Custom Argument Variables*.



From here, click the “Global” tab and navigate to the directory where the custom\_argvars file is stored (by default in the tools/iar/ directory of the installation path). Note that there are separate sets of custom\_argvars for the supported host devices.



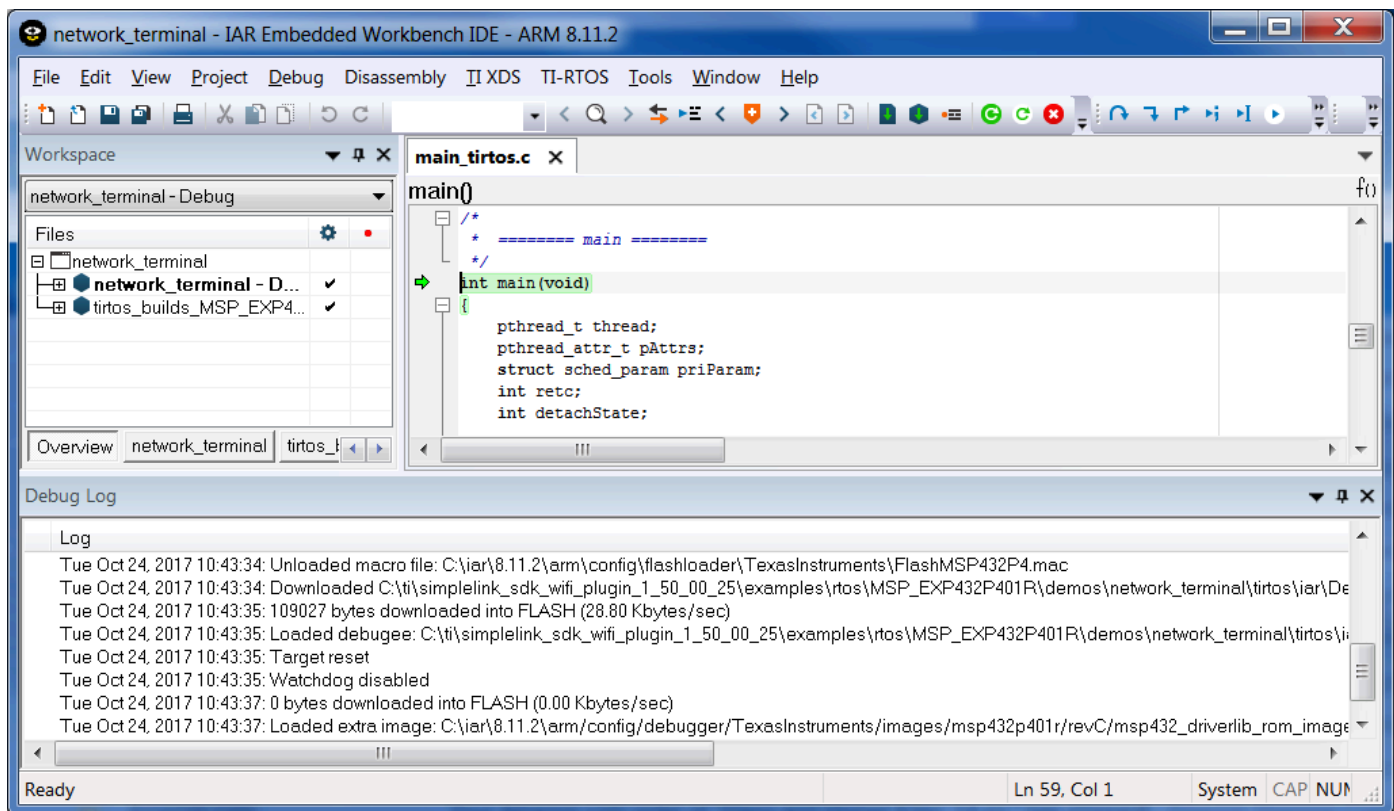
After importing this file the custom variables will show up in the text box in the middle of the screen. Click OK to closeout and save the dialog. Note that these custom variables need only be imported once and the settings will persist.



Once the external symbols have been imported you can either open the project manually on the file system or use IAR's built-in example project explorer. To open the project manually, simply go to **File->Open->Workspace**: Navigate to the example project's eww file. For example, the network terminal's eww file would be located at:

```
install_directory/examples/rtos/MSP_EXP432P401R/demos/network_terminal/tirtos/iar/network_terminal
```

Once you have opened the workspace in IAR you are able to download and debug the program as you would any other IAR project.



Note that if you upgrade or change versions of the SimpleLink SDK Wi-Fi Plugin or the base SDK, the external tools variables will have to change to point to the correct version of the software.

## Rebuilding the SimpleLink Wi-Fi Host Driver

While the SimpleLink Wi-Fi Host Driver linked to compiled binaries through simplelink.a is provided as a compiled binary with the plugin, you can rebuild it if needed. This is particularly relevant in case the host driver is ported to another platform with its own set of porting files. Rebuilding the host driver can be done in CCS, IAR, and GCC directly.

### Prerequisites

The host driver build relies on several resources that need to be downloaded and installed. In addition to the prerequisites mentioned in [\[section 2 of the quick start guide\]Quick\\_Start\\_Guide.html](#), one of the supported SimpleLink SDKs is required. Once those required prerequisites are installed, you must update the imports.mak file located at the root of the install directory to provide the paths to those installed resources.

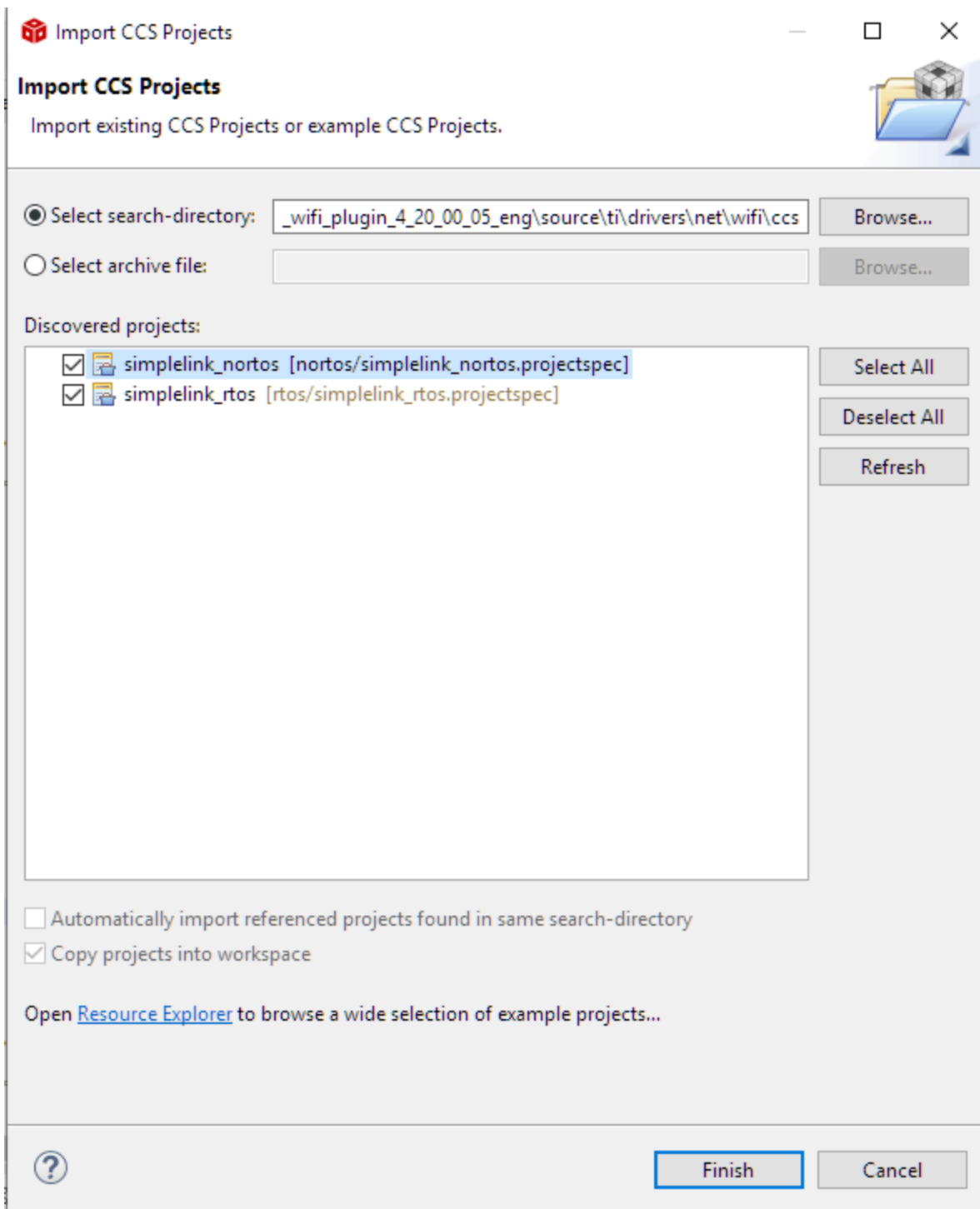
```

1 #
2 # Set location of various cgttools
3 #
4 # These variables can be set here or on the command line.
5 #
6 # The CCS_ARMCOMPILER, GCC_ARMCOMPILER, and IAR_ARMCOMPILER variables,
7 # in addition to pointing to their respective locations, also serve
8 # as "switches" for disabling a build using those cgttools. To disable a
9 # build using a specific cgttool, either set the cgttool's variable to
10 # empty or delete/comment-out its definition:
11 #     IAR_ARMCOMPILER ?=
12 # or
13 #     #IAR_ARMCOMPILER ?= c:/Program Files (x86)/IAR Systems/Embedded Workbench 7.5/arm
14 #
15 # If a cgttool's *_ARMCOMPILER variable is set (non-empty), various sub-makes
16 # in the installation will attempt to build with that cgttool. This means
17 # that if multiple *_ARMCOMPILER cgttool variables are set, the sub-makes
18 # will build using each non-empty *_ARMCOMPILER cgttool.
19 #
20 XDC_INSTALL_DIR      ?= c:/ti/xdctools_3_61_01_25_core
21
22 FREERTOS_INSTALL_DIR ?= c:/FreeRTOSv10.2.1_191129
23
24 SIMPLELINK_MSP432_SDK_INSTALL_DIR ?= c:/ti/simplelink_msp432p4_sdk_3_40_01_02
25 SIMPLELINK_MSP432E4_SDK_INSTALL_DIR ?= c:/ti/simplelink_msp432e4_sdk_4_20_00_11_eng
26 SIMPLELINK_CC13X2_26X2_SDK_INSTALL_DIR ?= c:/ti/simplelink_cc13x2_26x2_sdk_4_20_00_20_eng
27
28 CCS_ARMCOMPILER      ?= c:/ti/ccs1010/ccs/tools/compiler/ti-cgt-arm_20.2.1.LTS
29 GCC_ARMCOMPILER      ?= c:/ti/ccs1010/ccs/tools/compiler/gcc-arm-none-eabi-9-2019-q4-major-win32
30 IAR_ARMCOMPILER      ?= c:/Program Files (x86)/IAR Systems/Embedded Workbench 8.4/arm
31

```

## Building with CCS

To build the host driver, simply import the simplelink\_rtos and simplelink\_nortos CCS projects from the source/ti/drivers/net/wifi/ccs folder.



From there, rebuild the projects as you would a CCS example. They should rebuild without issues. If errors are encountered, please double-check your imports.mak and ensure that the paths provided are correct.

## Building with GCC and IAR

To build with GCC, you will need to use the gmake utility provided with your XDCtools install. The included makefile in source/ti/drivers/net/wifi/ will batch build the simplelink.a binary with GCC, IAR, as well as CCS. Before starting, ensure that your imports.mak file has the correct paths to CCS, GCC, as well as IAR as per the screenshot above.

On windows, you can open a command prompt in the source/ti/drivers/net/wifi/ directory, and then run /gmake.exe:

```
C:\ti\simplelink_sdk_wifi_plugin_4_20_00_05_eng\source\ti\drivers\net\wifi>C:\ti\xdctools_3_61_01_25_core\gmake.exe
gmake[1]: Entering directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/ccs/rtos'
gmake[1]: Nothing to be done for 'all'.
gmake[1]: Leaving directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/ccs/rtos'
gmake[1]: Entering directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/ccs/nortos'
gmake[1]: Nothing to be done for 'all'.
gmake[1]: Leaving directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/ccs/nortos'
gmake[1]: Entering directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/gcc/rtos'
gmake[1]: 'simplelink.a' is up to date.
gmake[1]: Leaving directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/gcc/rtos'
gmake[1]: Entering directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/gcc/nortos'
gmake[1]: 'simplelink.a' is up to date.
gmake[1]: Leaving directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/gcc/nortos'
gmake[1]: Entering directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/iar/rtos'
cc device.obj ...
cc driver.obj ...

        int Count = 0;
        ^
"C:\ti\simplelink_sdk_wifi_plugin_4_20_00_05_eng\source\ti\drivers\net\wifi\source\driver.c",1326  Warning[Pe177]:
        variable "Count" was declared but never referenced
cc flowcont.obj ...
cc fs.obj ...
cc netapp.obj ...
cc netcfg.obj ...
cc netutil.obj ...
cc nonos.obj ...
cc sl_socket.obj ...
cc spawn.obj ...
cc wlan.obj ...
cc cc_pal.obj ...
cc eventreg.obj ...
cc wlanconfig.obj ...
ar simplelink.a ...
gmake[1]: Leaving directory 'C:/ti/simplelink_sdk_wifi_plugin_4_20_00_05_eng/source/ti/drivers/net/wifi/iar/rtos'
```

That will invoke gmake on the Makefile and batch build the simplelink.a binaries for the GCC and IAR toolchains.

## Support

Please post on the [MSP432 E2E Forum](#) for device support questions.

---

TI is a global semiconductor design and manufacturing company. Innovate with 100,000+ analog ICs and embedded processors, along with software, tools and the industry's largest sales/support staff.

© Copyright 1995-2020, Texas Instruments Incorporated. All rights reserved.

[Trademarks](#) | [Privacy policy](#) | [Terms of use](#) | [Terms of sale](#)