SimpleLink MCU SDK用户指南

SimpleLink? MCU SDK介绍

SimpleLink? MCU Software Development Kit（SDK软件开发工具包）是一套软件开发工具，使工程师可以针对德州仪器公司（TI）的一系列微控制器开发应用程序。它是一个功能强大的软件工具包，通过对基本软件组件进行包装，易用的示例以及易用的软件开发包，提供了连贯一致的软件体验。

如果你还没有安装SDK及执行安装初始步骤，请参考【Quick Start Guide】位于安装目录【\ti\simplelink_cc32xx_sdk_1_30_01_03\docs\simplelink_mcu_sdk\Quick_Start_Guide.html】。

文档与支持

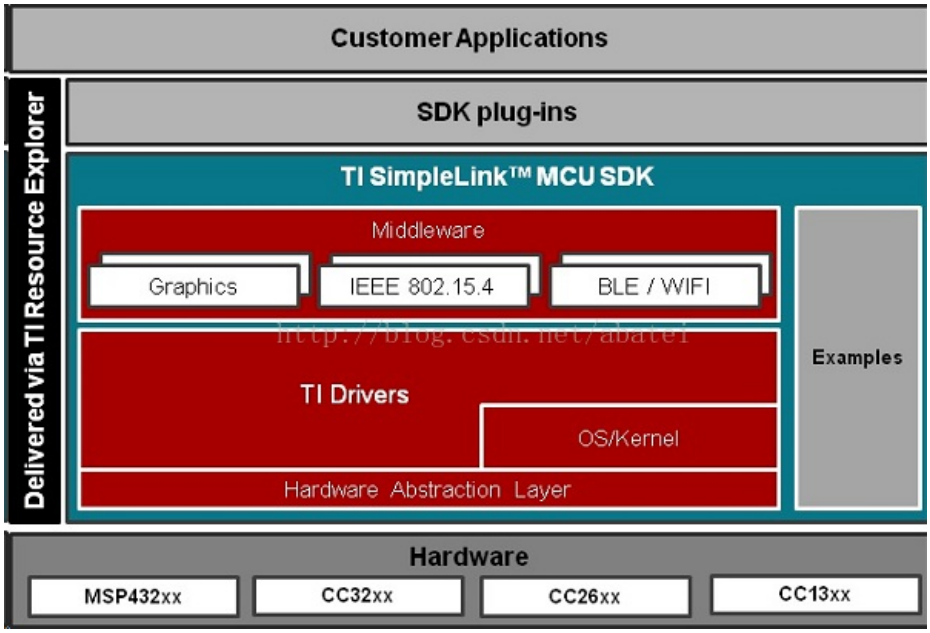
链接至SDK文档，其组件在【Documentation Overview】中提供，位于安装目录【ti\simplelink_cc32xx_sdk_1_30_01_03\docs\Documentation_Overview.html】

SimpleLink Academy提供了循序渐进的讨论及视频教程来介绍SimpleLink SDK的组件。

TI E2E Community支持此SDK。

SDK组件

SDK组件一起用于构建应用程序，SDK组件相互之间的关联如下图所示：

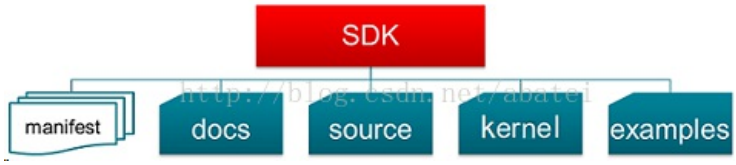


从架构图底部开始，组件如下：

- **Hardware**：TI SimpleLink系列：
  - MSP432：围绕ARM Cortex M4内核构建的低功耗MCU，为物联网传感器结点优化。
  - CC32xx：基于ARM Cortex M4 的MCU，整合了WiFi和安全功能。
  - CC26xx/CC13xx：基于ARM Cortex M3的低功耗无线MCU，用于高性能RF应用程序。它们支持一系列无线标准。
- **Hardware Abstraction Layer (HAL 硬件抽象层)**：抽象写硬件寄存器的C函数。这是驱动和操作系统内核用来访问硬件功能的层。
- **OS/Kernel（操作系统/内核）**：内核提供定时器和任务调试等服务。所有TI SimpleLink SDK都支持TI-RTOS内核。一些SDK还支持其它RTOS内核（如FreeRTOS）。
  - Driver Porting Layer (DPL驱动程序移植层)抽象了驱动接口。驱动程序使用RTOS功能如时钟、中断、互斥量及信号量。通过抽象这些功能，应用程序使用TI驱动时无需依赖于TI-RTOS，而可使用FreeRTOS内核代替。
  - POSIX层抽象了应用程序使用的RTOS内核的功能性。POSIX层使得示例及用户应用程序很容易地移植到不同的内核。使用此层是可选项（参考：“Choosing Whether to Use POSIX”章节）。
- **TI Driver API**：所有TI SimpleLink设备中具有相同硬件驱动程序设备的接口。虽然在不同芯片上UART的硬件实现有所不同，但TI Driver API使用同样的方法访问它们的公共功能。
- **Middleware（中间件）**：在驱动程序的基础上添加功能，例如通信栈和图形库。
- **Examples（示例）**：SDK提供了广泛的例子。它们的目的是让开发者更容易开始编写应用程序。每个示例都有自己的文档和项目文件。示例提供了使用RTOS的内核的方法，也提供了不使用RTOS的方法。

目录结构

SDK安装在/ti或默认的c:\ti。这也是所有与SDK一起安装的插件的安装位置。这是SDK安装的顶层目录结构：



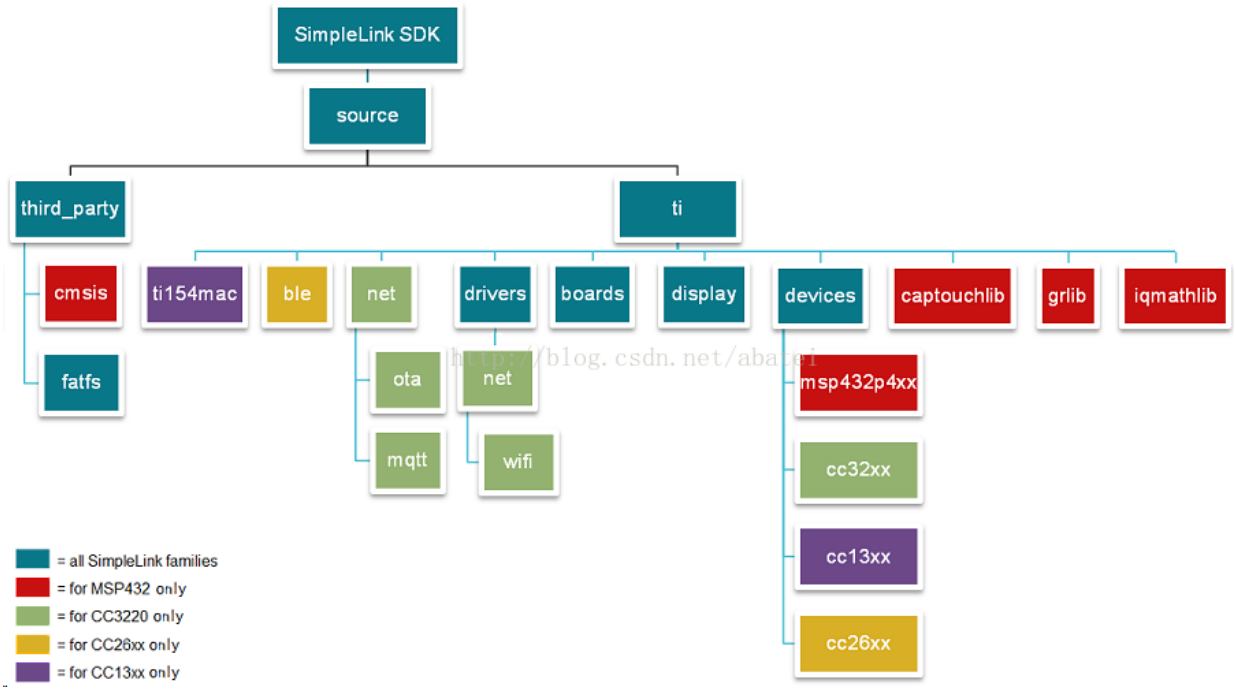
- “doc”目录包含了所有各种SDK组件的文档文件。

【file:///C:/ti/simplelink_cc32xx_sdk_1_30_01_03/docs/Documentation_Overview.html】里有各文档的链接。

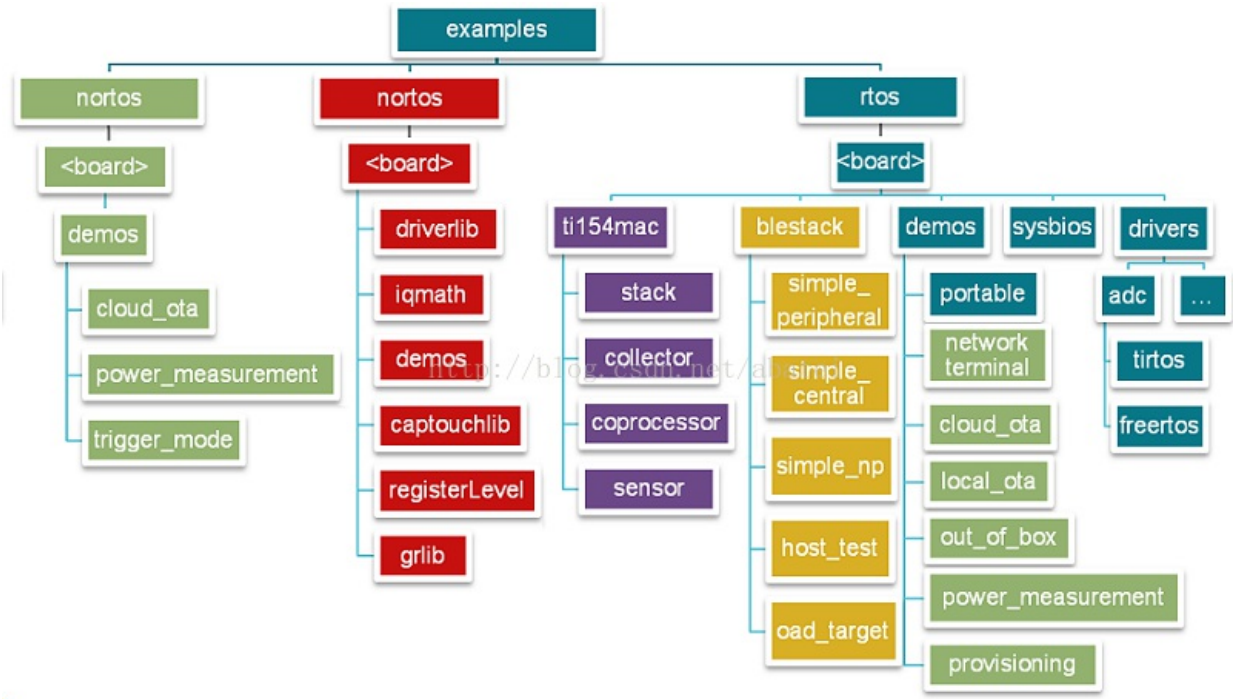
- “examples”目录包含了针对每个所支持开发板的示例应用程序。
- “kernel”目录包含了使用TI-RTOS内核以及FreeRTOS内核所需的文件。内核实现了DPL及POSIX层，使得SDK应用程序可以在不同内核间切换。
- “source”目录包含源码、库及编译应用程序时所使用的链接文件。这些包括驱动、主板、设备、中间件以及其它文件。例如，所有TI设备源码都在此提供。参考下图获取更多此目录树内容。

由于TI SimpleLink系列的不同SDK所使用的目录结构相同，有可能在不同设备间移动应用程序代码无需明显改动文件结构或包含路径。

下图显示了“source”目录的内容。注意左下角方块，显示了SDK中的特定目标所代表的颜色。



下图详细显示了“examples”目录所包含的内容，并使用了同样的颜色



未包含的工具和实用程序

SDK并不包含IDE或代码生成器，如编译器及连接器。你应该有一个可以支持的代码生成工包。合适的工具如TI代码生成工具Code Composer Studio，IAR Workbeacn及它的编译器和GNU编译器集合（gcc）。

SDK未安装FreeRTOS。你右通过FreeRTOS网站下载FreeRTOS源文件。SDK已经安装了Driver Porting Layer (DPL) 以及POSIX抽象使得FreeRTOS可与其它SDK组件一起使用。

SDK已经安装XDCtools，它包含了TI-RTOS内核的配置工具。XDCtools组件下面的工具可用于TI-RTOS及基内核的工具和模块。当你安装了SDK，XDCtools已经被装在了与SDK安装目录同级的平等目录中。

一些版本的SDK还提供了特定功能的工具。如用于蓝牙低功耗栈PC端接口的BTool。

理解一个SimpleLink MUC SDK应用程序

SimpleLink MUC SDK提供了一个叫demos\portable的例子，作为使用SDK创建你自己的应用程序的介绍。此示例使用了UART、I2C驱动及低功耗状态。它展示了多任务I/O驱动。此代码使用了POSIX，所以它很容易使用TI-RTOS内核及FreeRTOS内核运行。

应用程序有如下功能：

- 在阻塞的读写模式中初始化UART驱动。
- consoleThread线程提供了一个简单的控制台。当UART关闭，驱动可进入低功耗状态。
- temperatureThread线程通过I2C驱动从外设读取温度。

portable示例位于【<SDK_INSTALL_DIR>\examples\rtos\<board>\demos\portable】目录。它可用于所有支持的开发板、RTOS内核、工具链及IDE。

导入、编译并运行示例

如Quick Start Guide（位于file:///C:/ti/simplelink_cc32xx_sdk_1_30_01_03/docs/simplelink_mcu_sdk/Quick_Start_Guide.html）所描述的那样导入示例。分别介绍了在Code Composer Studio (CCS) IDE、IAR Embedded Workbench、GCC编译器的makefiles以及其它支持的开始环境中的使用情况。按照指南中的“Execute your First Application”这一节进行操作。

在Quick Start Guide中的相同章节所述编译“portable”。

在你导入的示例文件中有一个README.html文件。阅读此文件，按照介绍中的“Example Usage”节运行示例。

对于非CC3220开发板，还需要Sensors BoosterPack，它包含了一个TMP007温度传感器。

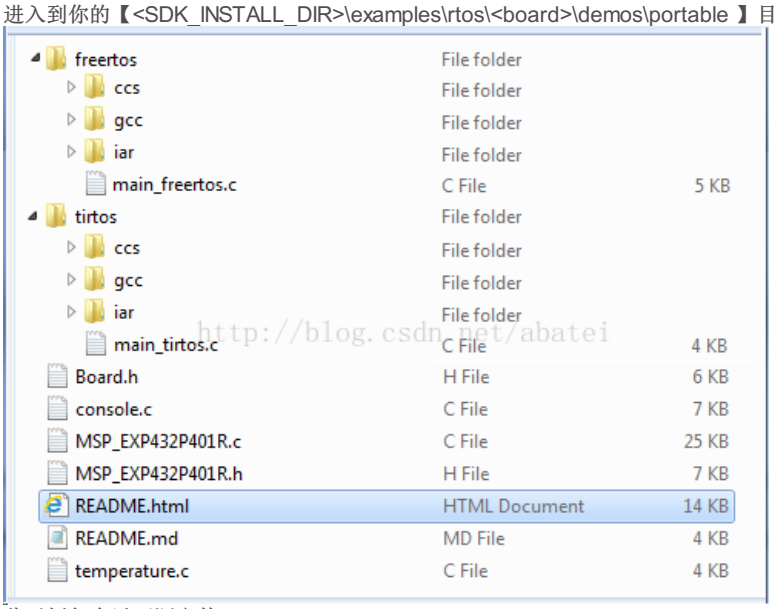
对于使用CCS的CC3220开发板，让设备“Free Run”而不是“Run”。

理解示例

此节审视“protable”示例代码，以理解它是如何创建的。

要理解示例是如何支持多个IDE及RTOS内核的，最好的方法是读SDK安装的示例文件。导入一个示例及删除文件的过程用于支持其它IDE及RTOS内核。

进入到你的【<SDK_INSTALL_DIR>\examples\rtos\<board>\demos\portable 】目录。



此示例包含以下源文件：

- main_tirtos.c和main_freertos.c：这些文件包含了main()函数。它们位于子目录，并使用POSIX（pthread）进行驱动程序的初始化和创建运行consoleThread及temperatureThread的线程。
- console.c：此文件包含由consoleThread线程所运行的函数，以及当主板BUTTON1按下时所产生的GPIO中断回调函数。
- temperature.c：此文件包含由temperatureThread线程所运行的函数。
- <board>.c和<board>.h（其中<board>是主板的名称）：这些文件设置特定主板项，如各种TI驱动的属性结构。例如，对于UART驱动，时钟源和引脚的使用进行配置。
- Board.h：将特定主板常量映射至可被应用程序使用的主板无关常量，从而使得应用程序代码具有可移植性。

以下章节更为详尽地描述了应用程序的这些部分。在你阅读这些章节时最好打开代码文件，以便更好理解应用程序是如何使用SDK的。

main_tirtos.c和main_freertos.c

两个版本的main()函数功能大多相同。因为POSIX的Pthreads用于代替内核API的直接调用，运行TI-RTOS内核及FreeRTOS内核只有轻微不同。

Header files

两个版本的文件都包含以下头文件：

- stdint.h
- pthread.h
- ti/drivers/GPIO.h（这是一个TI-RTOS包路径，位于<SDK_INSTALL_DIR/source）
- Board.h

FreeRTOS版本文件还包含FreeRTOS.h及task.h。

TI-RTOS版本文件还包含ti/sysbios/BIOS.h，此例中，位于<SDK_INSTALL_DIR>/kernel/tirtos/packages/ti/sysbios。

main()函数

1. main()函数开始于Board_initGeneral()的调用，此函数定义于Board.h文件，指向<board>.c文件中的一个特定板函数。

2. 接下来在pAttrs结构体设置各种线程的属性。

- 使用PTHREAD_CREATE_DETACHED状态调用pthread_attr_setdetachstate()，使得创建的线程处于分离状态，因为此应用程序所使用的线程永远无需连接其它线程。
- 调用pthread_attr_setschedparam()将线程调度优先级设置为1（最小优先）。
- 调用pthread_attr_setstacksize()将线程所使用的栈尺寸设置为THREADSTACKSIZE，它在相同文件前面定义。

3. 接下来main()函数两次调用pthread_create()。

- 第一次创建了一个将要运行consoleThread()函数的线程。此线程运行于默认优先级（1，最小值）。（参考console.c文件获知此线程所执行的动作）
- 第二次创建了一个将要运行temperatureThread()函数的线程。此线程的优先级高于consoleThread（优先级2）。这样，如果两个线程都准备运行，temperatureThread则优先运行。当temperature线程阻塞或从I2C等待数据时，console线程得以运行。（参考temperature.c文件获知此线程所执行的动作）

例如：

[cpp] view plain copy

```
priParam.sched_priority = 2;
pthread_attr_setschedparam(&pAttrs, &priParam);

retc = pthread_create(&thread, &pAttrs, temperatureThread, NULL);
if (retc != 0)
{
    /* pthread_create() failed */
    while (1);
}
```

参考wiki主题SYS/BIOS POSIX Thread (pthread) Support获取有关SDK支持哪些POSIX API的更详细信息。

如果你的应用程序直接选择TI-RTOS内核而不是POSIX，则直接调用Task_create() API而不是pthread_create() API：

[cpp] view plain copy

```
Task_Params_init(&taskParams);
taskParams.priority = 2;
taskParams.stackSize = 768;

temperatureTaskHandle = Task_create(temperatureThread, &taskParams, &eb);
```

参考 TI-RTOS Kernel User's Guide获取更多有关TI-RTOS内核API的信息。

1. main()函数接下来创建一个互斥体，将用于 consoleThread() 及temperatureThread()更新温度时护它的安全。
2. 接下来调用GPIO_init()来初始化GPIO驱动。此项工作在main()中完成是因为线程将使用GPIO驱动。
3. 此时，两个版本的main()函数分开。

-
  - TI-RTOS版本内核简单调用BIOS_start()以开始调度。BIOS_start()不会返回。
  - FreeRTOS版本调用vTaskStartScheduler()以开始FreeRTOS调度。vTaskStartScheduler()不会返回。

console.c

此文件对于所有开发板、RTOS内核及开发环境是相同的。记得查看README.html文件获取有关如何打开一个串口会话来运行示例控制台的内容。

consoleThread()函数在RTOS开始调度时运行。首先它会执行一些配置任务，接下来进入while循环，直到应用程序停止。

在函数的配置部分，consoleThread()函数执行以下动作：

1. 使能CC3220开发板的电源管理，为了方便调试，CC3220默认情况下是关掉电源管理的。
2. 配置Board_GPIO_BUTTON1在回应按键时运行gpioButtonFxn()以唤醒主板。
3. 调用POSIX函数sem_init()以初始化一个匿名信号量。按钮按下将提交信号量，之后的while循环则等待信号量。
4. 初始化UART通信参数。

在while循环中，consoleThread()函数执行以下动作：

1. 如果uartEnabled为false（当控制台关闭时发生），等待信号量，直到按钮按下。
2. 通过配置参数调用UART_open()在阻塞读写模式下为控制台打开一个UART。
3. 运行simpleConsole()函数，它使用UART_write()和UART_read()来管理控制台的用户接口，此用户接口接收以下命令：

[cpp] view plain copy

```
Valid Commands
--------------
h: help
q: quit and shutdown UART
c: clear the screen
t: display current temperature
```

1. 如果使用“t”命令，由temperatureThread所提供的外部变量temperatureC和temperatureF的值都存储于本地。一个互斥量用于保护这些操作。没有互斥量，拥有更高优先级的temperature线程可能会中断console线程对这些变量的读取。这会导致console线程所打印的摄氏温度值与华氏温度值不匹配。
2. 静态itoa()函数用于将整数温度值转换为字符串，UART_write()用于将值输出到控制台。
3. 如果使用“q”命令，simpleConsole()函数返回并调用UART_close()。UART关闭后，电源管理自动进入低功耗状态。参考TI-RTOS Power Management User’s Guide获取更多有关电源状态的信息。

temperature.c

此文件对于所有开发板、RTOS内核及开发环境是相同的。

temperatureThread()函数在RTOS开始调度时运行。首先它会执行一些配置任务，然后进入while循环直到应用程序关闭。线程在while循环结尾处阻塞等待semTimer的信号量。定时器每隔1秒提交一次semTimer信号量以让任务解除阻塞。sleep()函数可以使用，但会发生一个很小的偏移量。

在函数的配置部分，temperatureThread()函数执行以下动作：

1. 初始化I2C驱动并打开驱动使用。
2. 初始化I2C_transaction结构体，它将用于I2C_transfer()。
3. 初始化定时器和semTimer信号量。

在while循环中，temperatureThread()函数执行这些动作：

1. 如果I2C传输成功，则互斥量锁止以让随后的线程操作安全。
2. 从接收到的数据提取温度。（对于C3200开发板，默认读取板载TMP006传感器的TMP_DIE_TEMP寄存器。对于所有其它开发板，示例读取Sensors BoosterPack上的TMP007传感器的TMMP_OBJ_TEMP寄存器。）
3. 温度在摄氏温度下被精确化，并转化为华氏温度。两个值都被存储并输出。
4. 释放互斥锁。
5. 如果温度超过30度，将使用GPIO_write()写一个警告信息，否则，所有之前的警告信息被清空。
6. while循环阻塞于semTimer信号量，此信号量由线程创建的定时器每秒提交一次，这为设备提供了足够的时间在电源管理的控制下转换为低功耗状态。

注意：MSP432使用看门狗以实现更好地能量节省，详情请参考wiki主题TI-RTOS MSP432 Timer。

<board>.c和<board>.h

（<board>是你的开发板的名称）这些文件负责建立开发板的主板特定项。例如，GPIO部分配置GPIO输入、输出引脚和LED。它为输入引脚创建了一组回调函数并设置为NULL。对于每个驱动，它都声明了一个配置结构体并将结构体内属性字段设置为默认值。

<board>.c文件还配置TI驱动使用的结构体。这些结构体对于大多数TI设备相同。例如，<board>.c中定义的一个叫`<Driver>_Config`的配置数据结构体。每个驱动的配置包含一个指向函数表的指针，一个指向对象的指针，和一个指向一组硬件属性设置的指针。有关驱动配置结构体的更详细信息，参考detailed reference information，位于
【file:///C:/ti/simplelink_cc32xx_sdk_1_30_01_03/docs/tidrivers/doxygen/html/index.html】。

Board.h

此文件将主板特定常量和函数映射至主板无关常及函数。主板无关名称将被应用程序使用，从而应用程序代码变得可移植。

CCS、GCC和IA文件
对于每个支持的RTOS，当你在开发环境导入示例时，它包含了为此示例建立项目的文件。其中包含 Composer Studio (CCS)、IAR Embedded Workbench和 the GNU Compiler Collection (GCC) 的代码文件。