SMeshStudio 快速入门指南

傅骞 博士 fuqian@smeshlink.com

1 SMeshStudio 简介

当然物联网开发领域流行着三大可商用的开源软件平台，它们分别是 Arduino, mBed 和 Contiki，它们在各自的应用领域都有着自己的优势，简单来说，Arduino 重在简单易用，可以方便地连接各类传感器；mBed 重在支持广泛，应用开发通用性强；Contiki 重在网络功能强大，应用开发互联方便。但对于开发者来说，这三者都没有提供让开发者满意的本地化 IDE 开发环境，如 Arduino 的开发环境过于简单，mBed 只提供在线开发环境，Contiki 干脆全部用命令行操作，这在一定程度上降低了开发者的开发效率，同时也减缓了这三者的推广进度。

SMeshStudio 就是在这样的背景下产生的，它基于 Eclipse 和 Arduino Eclipse Plugin 开发（感谢他们做出的伟大产品），支持 Arduino, mBed 和 Contiki 应用的开发、编译和上载（不支持调试），可以大大加快开发者使用上述开源系统进行应用开发的过程，具体来说，SMeshStudio 具有以下特点：

- 免安装，免配置，解压后就能直接使用。SMeshStudio 全部采用 Java 编写，并在内部集成了用户开发所需的编译器、上载工具和各类源代码库，所以只要用户计算机中已经有了 java 运行环境，下载后解压就能直接使用。考虑到 64 位的逐步普及，SMeshStudio 有 32 位和 64 位两个版本可以选择。
- 多平台支持。SMeshStudio 可以支持多个平台的开发，在软件上包括 contiki，Arduio 和 mbed，在硬件上可以支持各类采用 gcc 编译的微处理器平台，主要是 AVR 和 ARM，系统会根据用户的选择自动载入相应的代码和编译器。
- 向导式项目创建。SMeshStudio 提供了项目创建向导，用户只要根据向导完成项目类型、项目名称、开发板类型、程序上载端口的选择，SMeshStudio 就会自动创建好相应项目的模板。
- 集成 Eclipse 强大 IDE 功能。Eclipse 提供了强大的 IDE 功能，其中最常用的有查看函数申明、格式化代码、自动方法提示等。
- 图形化上载。SMeshStudio 集成了多种程序上载工具，用户只要在向导中完成了正确的配置，就能采用图形化界面完成程序上载工作，省去了命令行操作。
- SMeshStudio 解压下会生成两个目录，一个是 eclipse，里面放的是增加了 plugin 后的 eclipse 系统；另外一个是 smeshcore，里面放的是各类编译器和开源软件库。用户使用 eclipse\smeshstuio.exe 启动，首次使用有可能产生网络访问警告，用户可以根据自己的需要自行选择，下图是 SMeshStudio 第一次启动后的欢迎页面（mBed 是 SMeshStudio 推荐的开发平台，所以直接链接到了 mBed 资料提供页面），用户关闭欢迎页面后就可以开始各类应用的开发过程：

图表 1-1 SMeshStudio 欢迎页面

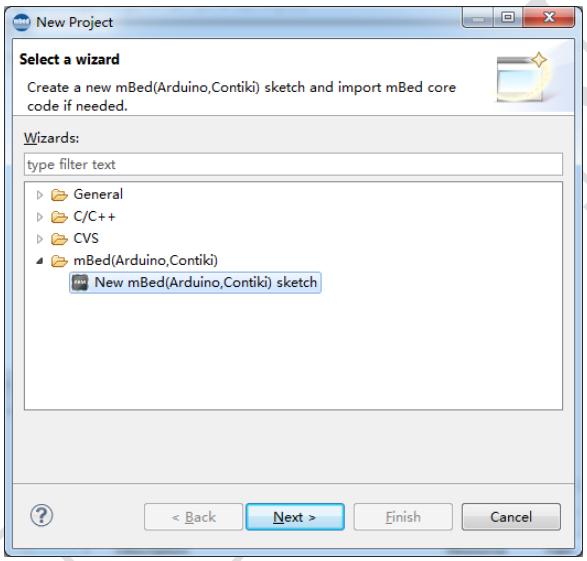
SMESHLINK

2 SMeshStudio Arduino 入门指南

2.1 SMeshStudio Arduino 快速入门

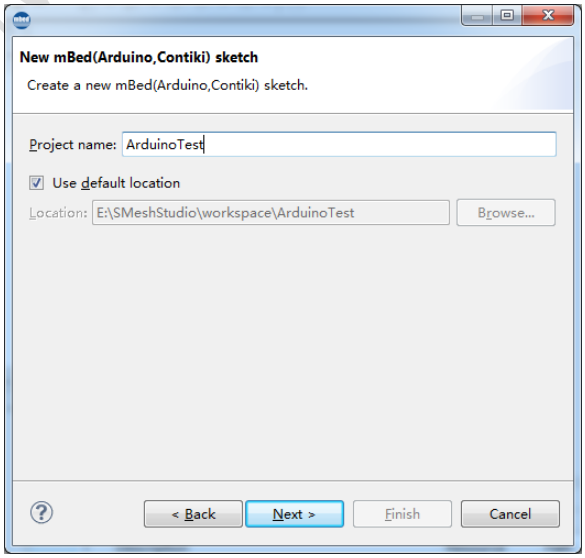
Arduino 是当前最流行的硬件开源系统，SMeshStudio 就是基于 Arduino 的 Eclipse 插件开发的，所以 SMeshStudio 可以很方便地完成基于 Arduino 核心库的应用开发。具体过程如下：

1. **选择 Eclipse 项目类型**：SMeshStudio 建立在标准 Eclipse 开发环境基础之上，它可以开发多种类型的应用程序，所以用户必须选择合适的开发类型，在 SMeshStudio 中，mBed, Arduino 和 Contiki 都属于同种类型。用户可以通过菜单 File——>New——>Projects 启动下面的项目类型选择界面并选择 New mBed(Arduino,Contiki) sketch 项目类型：



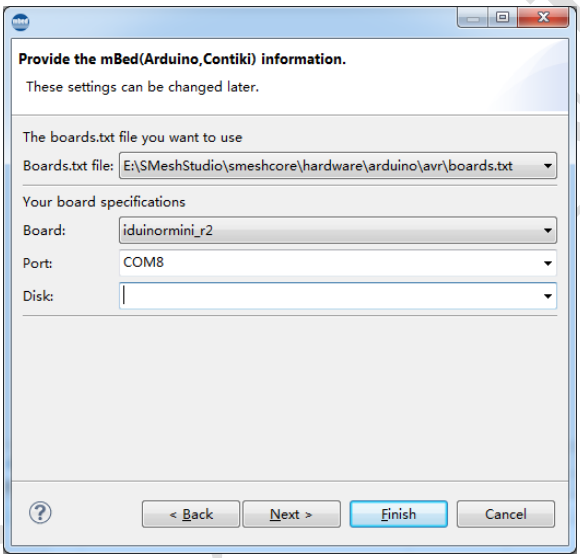
图表 2-1 SMeshStudio 项目类型选择

2. **设置项目名称**：用户选择 next 继续后出现项目名称设置界面，用户在这里可以随意设置喜欢的项目名称并设置项目保存位置（建议使用缺省者），用户在这里设置项目名称为 ArduinoTest：



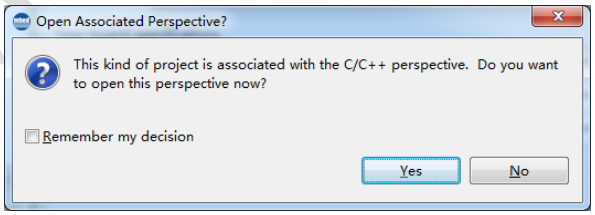
图表 2-2 SMeshStudio 设置项目名称

3. **选择开发板类型**：用户选择 next 继续后出现开发板选择界面。SMeshStudio 支持多种软件开发平台，每个软件开发平台下又可以支持多个开发板，为了让用户的项目能匹配上用户的开发板，这一步的选择就显得非常重要。SMeshStudio 开发板的软件匹配原则如下：SMeshStudio 只根据开发板名称匹配用户项目，如果开发板名称中包含 bed，SMeshStudio 会把项目识别成 mBed 项目；如果包含 contiki，SMeshStudio 会把项目识别成 contiki 项目，否则 SMeshStudio 会把项目识别成 Arduino 项目。为了方便用户的选择，SMeshStudio 已经把不同的开发板放到不同的开发板描述文件中，所有用户在这里首先要选择开发板文件，然后再选择具体的开发板，之后用户还需要设置开发板上载程序使用的串口，考虑到有些开发板可以采用文件复制方式上载，SMeshStudio 也提供了磁盘选择选项，串口号和磁盘盘符必须设置一项后才能继续。用户在这里可以选择 iduinomini_r2，它是一块基于 Atmega128RFA1 的 Arduino 开发板，同时也支持 Contiki 系统，是 SMeshStudio 推荐的开发板之一：



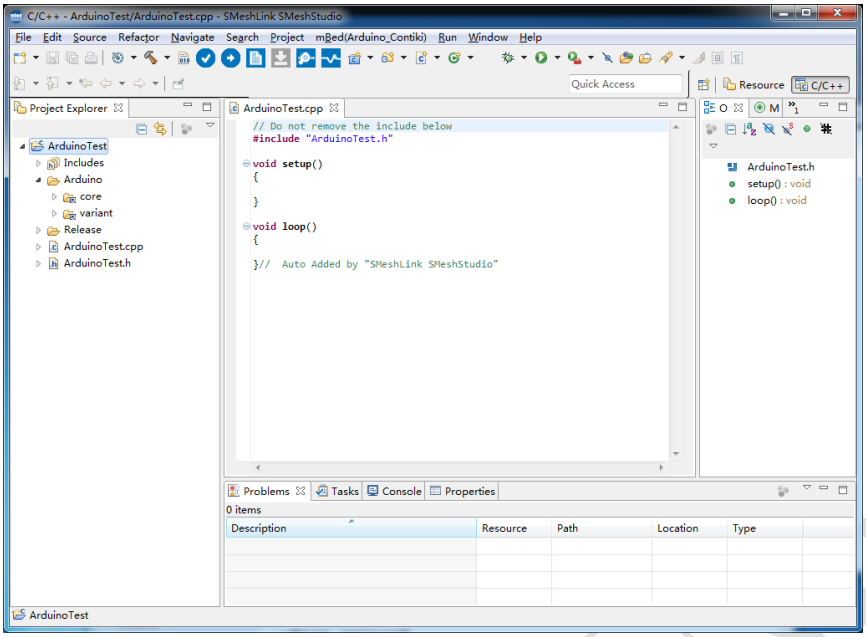
图表 2-3 SMeshStudio 选择开发板类型

4. **结束向导**：用户此时可以选择 Finish 结束向导，生成 Arduino 项目，此时系统有可能提示说这是一个 C/C++项目，建议选择使用 C/C++视图，用户直接选择 yes 并建议选择 Remember my decision：



图表 2-4 SMeshStudio 视图类型选择

这样一来，用户就完成了 Arduino 项目的创建过程，此时，SMeshStudio 界面如下：



图表 2-5 SMeshStudio 初始 Arduino 项目

5. **编写代码**：此刻，用户可以来简单地理解一下 ArduinoTest 项目，任何基于 SMeshStudio 的 Arduino 项目的代码都由三部分组成，具体列表如下：

- Arduino 核心库，即 Project Explorer 视图中的 Arduino 目录，该目录下有两个文件夹，一个是 core，里面放的是 Arduino 统一的核心库；另外一个是 variant，里面放的是 Arduino 特定板卡的管脚定义文件 pins_arduino.h。
- Libraries 扩展库，即 Project Explorer 视图中的 Libraries 目录，里面存放着用户导入的和特定应用相关的扩展库，该目录只有在用户导入扩展库后才会存在，ArduinoTest 没有导入任何扩展库，所以不存在该目录。
- 用户项目代码，向导默认会生成两个文件，一个是和项目同名的 CPP 文件，另外一个则是和项目同名的.h 头文件。

下面编写一个最简单的 Arduino 程序，其作用是每一秒输出一个 Hello World!并变换 LED，其代码如下：

```
#include "ArduinoTest.h"
void setup()
{
  Serial.begin(38400);
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
  pinMode(LED3,OUTPUT);
}
  void loop()
{
   delay(1000);
   Serial.println("Hello World!");
   digitalWrite(LED1,!digitalRead(LED1));
   digitalWrite(LED2,!digitalRead(LED2));
   digitalWrite(LED3,!digitalRead(LED3));
}
```

6. **编译项目**：用户代码编写完毕后就可以使用 Project——>Build Project 编译代码，如

果没有错误的话，用户就可以在控制台看到如下内容，它表示用户程序的内存使用情况，其数值只要不超过 100%就可以了。

Building target: ArduinoTest
Printing size:
"E:/SMeshStudio/smeshcore/tools/avr/bin/avr-size" --format=avr --mcu=atmega128rfa1 "E:/SMeshStudio/workspace/ArduinoTest/Release/ArduinoTest.elf"
AVR Memory Usage

Device: atmega128rfa1

Program: 5258 bytes (4.0% Full)
(.text + .data + .bootloader)

Data: 1182 bytes (7.2% Full)
(.data + .bss + .noinit)

Finished building target: ArduinoTest

16:24:13 Build Finished (took 232ms)

7. **上载程序：**用户编译成功后就可以使用 mbed(Arduino_Contiki)——>upload sketch 上载程序，如果没有问题的话，用户可以看到下面的输出结果，至此，基于 SMeshStudio 的简单 Arduino 程序开发完毕：

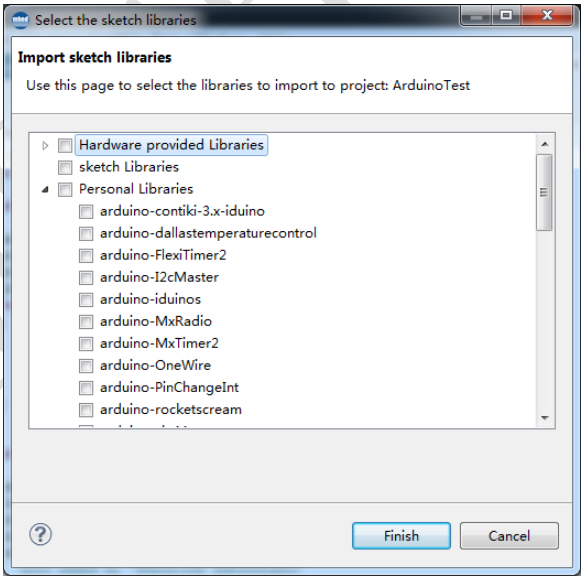
Starting upload
using mbed loader
Launching E:\SMeshStudio\smeshcore\tools\avr\bin\avrdude -CE:\SMeshStudio\smeshcore\tools\avr\bin\avrdude.conf -patmega128rfa1 -carduino -PCOM11 -b57600 -Uflash:w:E:\SMeshStudio\workspace\ArduinoTest\Release\ArduinoTest.hex:i
Output:
avrdude: AVR device initialized and ready to accept instructions
Reading | ## | 100% 0.00s
avrdude: Device signature = 0x1ea701
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "E:\SMeshStudio\workspace\ArduinoTest\Release\ArduinoTest.hex"
avrdude: writing flash (5306 bytes):
Writing | ## | 100% 1.27s
avrdude: 5306 bytes of flash written
avrdude: verifying flash memory against

SMESHLINK

```
E:\SMeshStudio\workspace\ArduinoTest\Release\ArduinoTest.hex:
avrdude: load data flash data from input file
E:\SMeshStudio\workspace\ArduinoTest\Release\ArduinoTest.hex:
avrdude: input file E:\SMeshStudio\workspace\ArduinoTest\Release\ArduinoTest.hex
contains 5306 bytes
avrdude: reading on-chip flash data:
Reading | ################################################## | 100% 0.99s
avrdude: verifying ...
avrdude: 5306 bytes of flash verified
avrdude done.  Thank you.
avrdude finished
upload done
```

2.2 SMeshStudio Arduino 扩展库应用

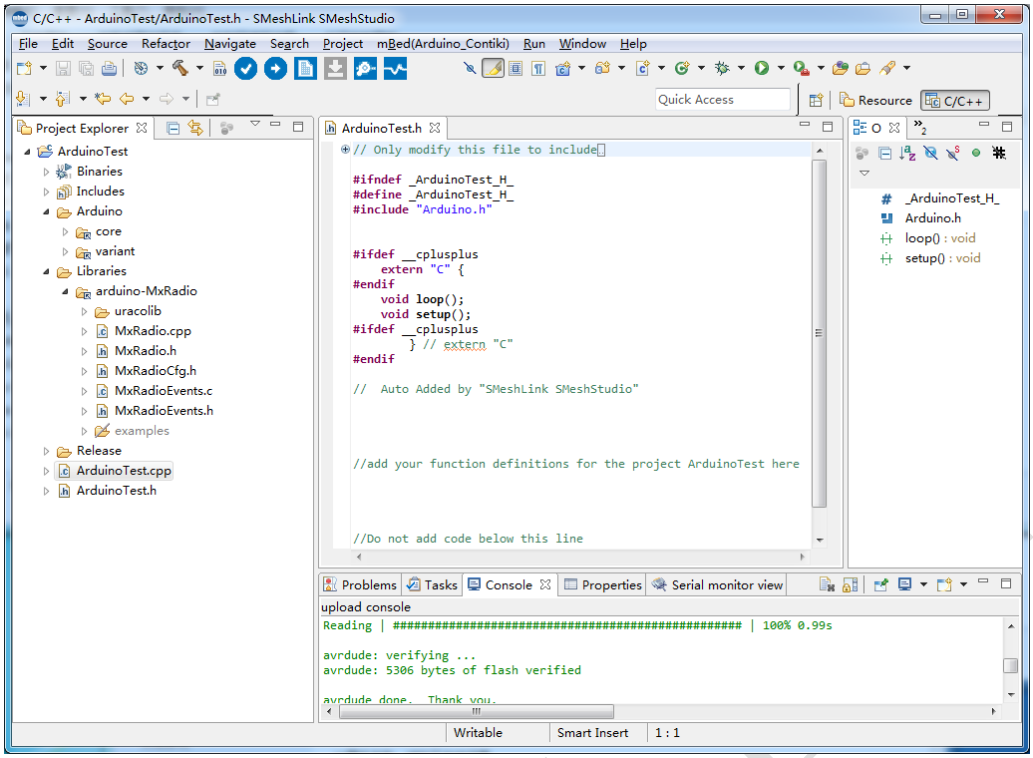
Arduino 官方和第三方提供了大量的扩展库，它们的使用可以大大简化用户应用的开发，这也是 Arduino 系统最吸引人的地方。SMeshStudio 提供的 Arduino 扩展库主要有两个部分，一个是 Arduino 官方提供的，另外一个是第三方提供的，当用户使用 mBed(Arduino_Contiki)——>Add a library to the selected project 菜单后就可以得到下面的界面。Arduino 官方提供的官方库在 Hardware provided Libraries 分类下，而第三方提供的包括 SMeshStudio 自带的则在 Personal Libraries 目录下：



图表 2-6 SMeshStudio 导入 Arduino 扩展库

需要注意的是，由于 SMeshStudio 支持多个软件平台，所以它在 Personal Libraries 中也存放着多个平台的扩展库，用户在开发基于 Arduino 的应用时只能使用 arduino 开头的扩展库，接下来用户需要导入 arduino-MxRadio 库，该库用来无线收发数据，导入后，ArduinoTest 项目的目录结构变化如下：

SMESHLINK



图表 2-7 SMeshStudio 导入扩展库目录变化

接下来，用户需要把 ArduioTest 代码变动如下：

```
#include "ArduinoTest.h"
#include "MxRadio.h"
uint8_t i;
void errHandle(radio_error_t err)
{
    digitalWrite(LED2,!digitalRead(LED2));
}
void onXmitDone(radio_tx_done_t x)
{
    digitalWrite(LED1,!digitalRead(LED1));
}
uint8_t* onReceiveFrame(uint8_t len, uint8_t* frm, uint8_t lqi, int8_t ed,uint8_t crc_fail)
{
    digitalWrite(LED3,!digitalRead(LED3));
    return frm;
}
void setup()
{
    MxRadio.begin(11);
    Serial.begin(9600);
    pinMode(LED1,OUTPUT);
    pinMode(LED2,OUTPUT);
    pinMode(LED3,OUTPUT);
    MxRadio.attachError(errHandle);
    MxRadio.attachTxDone(onXmitDone);
    MxRadio.attachReceiveFrame(onReceiveFrame);
```

SMESHLINK

```
}
void loop()
{
    MxRadio.beginTransmission();
    MxRadio.write("Hello World!");
    MxRadio.write(i);
    i++;
    MxRadio.endTransmission();
    delay(1000);
}
```

重复前面的程序上载过程，你就可以看到 LED1 每 1 秒钟变化一次，从而验证数据无线发送成功，当然也验证了 Arduino 扩展库的使用成功。

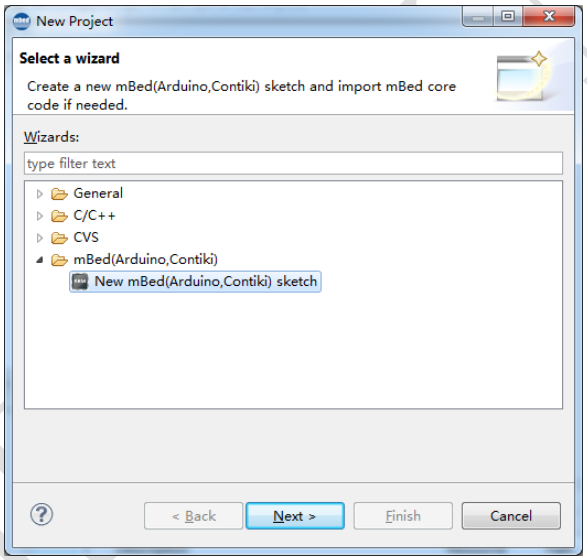
SMESHLINK

3 SMeshStudio Contiki 入门指南

3.1 SMeshStudio Contiki 快速入门

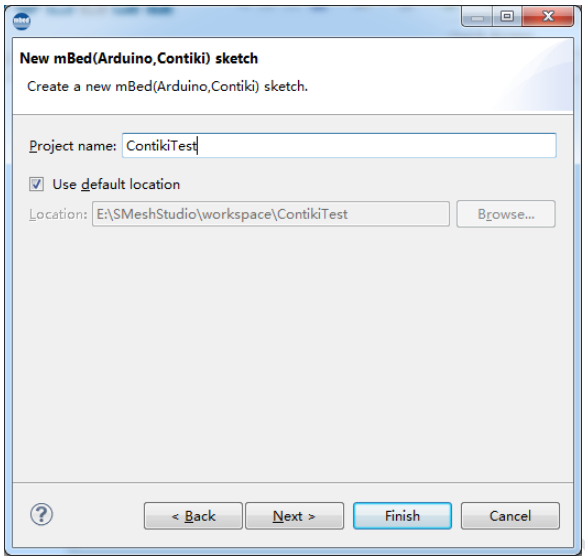
Contiki 是当前最流行的物联网操作系统之一，它具有非常强大的网络连接能力，支持 IPV4/IPv6，TCP，UDP，HTTP，COAP，DNS，RPL 等各类互联网标准协议，但 Contiki 原有的开发方式基于命令行方式，使用起来非常不便，所以，SMeshStudio 借鉴了 Arduino 的开发方式，实现了和 Arduino 一样方便的应用开发。具体过程如下（其实大部分过程是一样的，但考虑到有些人只关心 Arduino，mBed 和 Contiki 中的一个，所以这里还是完整地重复了一遍）：

1. **选择 Eclipse 项目类型**：SMeshStudio 建立在标准 Eclipse 开发环境基础之上，它可以开发多种类型的应用程序，所以用户必须选择合适的开发类型，在 SMeshStudio 中，mBed, Arduino 和 Contiki 都属于同种类型。用户可以通过菜单 File——>New——>Projects 启动下面的项目类型选择界面并选择 New mBed(Arduino,Contiki) sketch 项目类型：



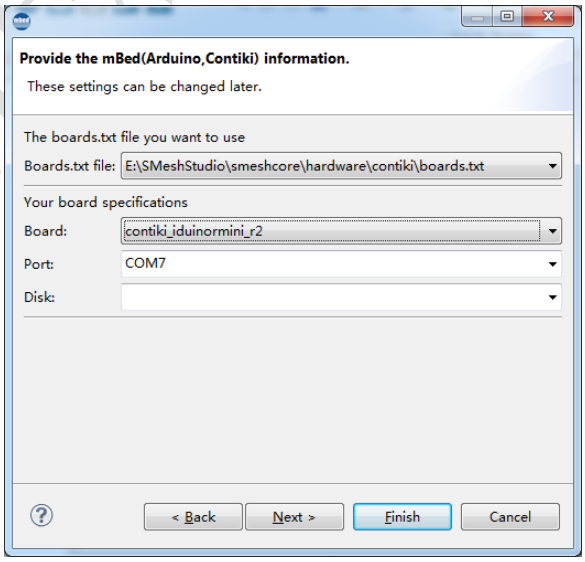
图表 3-1 SMeshStudio **项目类型选择**

2. **设置项目名称**：用户选择 next 继续后出现项目名称设置界面，用户在这里可以随意设置喜欢的项目名称并设置项目保存位置（建议使用缺省者），用户在这里设置项目名称为 ContikiTest：



图表 3-2 SMeshStudio 设置项目名称

3. **选择开发板类型**：用户选择 next 继续后出现开发板选择界面。SMeshStudio 支持多种软件开发平台，每个软件开发平台下又可以支持多个开发板，为了让用户的项目能匹配上用户的开发板，这一步的选择就显得非常重要。SMeshStudio 开发板的软件匹配原则如下：SMeshStudio 只根据开发板名称匹配用户项目，如果开发板名称中包含 bed，SMeshStudio 会把项目识别成 mBed 项目；如果包含 contiki，SMeshStudio 会把项目识别成 contiki 项目，否则 SMeshStudio 会把项目识别成 Arduino 项目。为了方便用户的选择，SMeshStudio 已经把不同的开发板放到不同的开发板描述文件中，所有用户在这里首先要选择开发板文件，然后再选择具体的开发板，之后用户还需要设置开发板上载程序使用的串口，考虑到有些开发板可以采用文件复制方式上载，SMeshStudio 也提供了磁盘选择选项，串口号和磁盘盘符必须设置一项后才能继续。用户可以选择 contiki_iduinomini_r2（它实际上就是 iduinomini_r2，但为了被识别成 Contiki，所以在前面加上了 contiki），它是一块基于 Atmega128RFA1 的 Arduino 开发板，同时也支持 Contiki 系统，是 SMeshStudio 推荐的开发板之一：

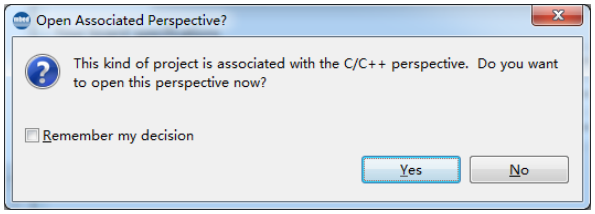


图表 3-3 SMeshStudio 选择开发板类型

4. **结束向导**：用户此时可以选择 Finish 结束向导，生成 Contiki 项目，此时系统有可能提示说这是一个 C/C++项目，建议选择使用 C/C++视图，用户直接选择 yes 并建议选择

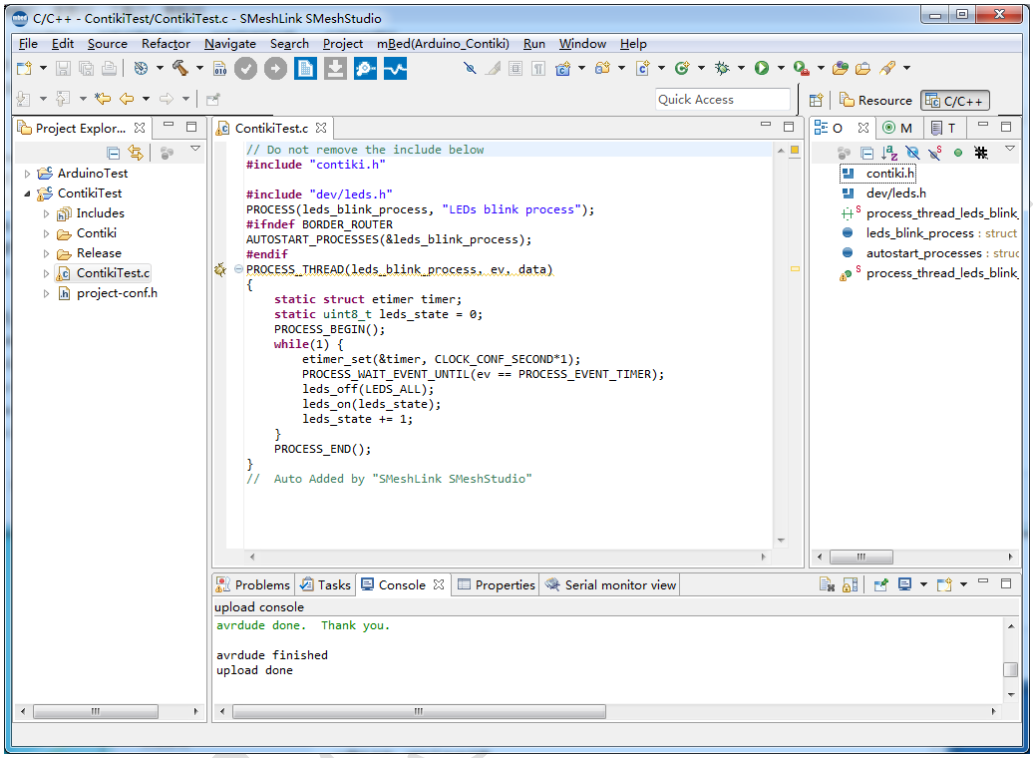
SMESHLINK

Remember my decision：



图表 3-4 SMeshStudio 视图类型选择

这样一来，用户就完成了 Contiki 项目的创建过程，此时，SMeshStudio 界面如下：



图表 3-5 SMeshStudio 初始 Contiki 项目

5. **编写代码**：此刻，用户可以来简单地理解一下 ContikiTest 项目，任何基于 SMeshStudio 的 Contiki 项目的代码都由三部分组成，具体列表如下：

- Contiki 核心库，即 Project Explorer 视图中的 Contiki 目录，该目录下有两个文件夹，一个是 core，里面放的是 contiki 独立于硬件部分的实现；另外一个是 variant，里面放的是 contiki 和硬件相关的实现，其中一个是 cpu，里面放的是和 cpu 相关的代码，另外一个是 platform，里面放的是和具体的开发板相关的代码。
- Libraries 扩展库，即 Project Explorer 视图中的 Libraries 目录，里面存放着用户导入的和特定应用相关的扩展库（由于本项目没有导入扩展库，所以本项目中并不存在该目录，但在接下来的例子中你会看到）。
- 用户项目代码，向导默认会生成两个文件，一个是名为 project-conf.h 的项目配置文件，另外一个则是和项目同名的 c 文件。

项目中的 project-conf.h 主要用来设置本项目无线网络所使用的频段和 mac 地址，具体介绍如下：

```
#ifndef _t_H_
#define _t_H_
//去掉编译器自动添加的ARDUINO宏定义
#undef ARDUINO
```

```
//设置节点的mac地址定义方式0表示使用ds2411自动设置，1表示使用下面的
EUI64_ADDRESS手动设置
#define UIP_CONF_EUI64    1
//设置节点的功耗模式，当前系统还不支持低功耗，所以这里只能设成1
#define LOWPOWER 0
//设置节点的mac地址，该定义只有在UIP_CONF_EUI64为1时有效
#define EUI64_ADDRESS {0x02, 0, 0, 0, 0, 0, 0, 0xbb};
// 设置节点的工作频段，范围是11-26
#define RFCHANNEL 26
//下面是在系统编译时会自动添加的宏定义
//#define AUTOSTART_ENABLE 1
//#define UIP_CONF_IPV6 1
//#define RF230BB 1
//#define WATCHDOG_CONF_TIMEOUT WDTO_8S
//#define AUTO_CRC_PADDING 2
//#define HAVE_STDINT_H
#endif /* _t_H_ */
```

而本项目中 ContikiTest.c 就是用户应用的唯一代码，它的作用是每一秒钟变换一次 LED，列表解释如下：

```
#include "contiki.h"
#include "dev/leds.h"
PROCESS(leds_blink_process, "LEDs blink process");
#ifndef BORDER_ROUTER    //如果是BORDER_ROUER，则系统会自动启动
BORDER_ROUER Process
AUTOSTART_PROCESSES(&leds_blink_process);
#endif
PROCESS_THREAD(leds_blink_process, ev, data)
{
    static struct etimer timer;
    static uint8_t leds_state = 0;
    PROCESS_BEGIN();
    while(1) {
        etimer_set(&timer, CLOCK_CONF_SECOND*1);
        PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER); //等待1
秒，这段时间别的Process有机会执行，体现多任务特点
        leds_off(LEDS_ALL); //熄灭所有灯
        leds_on(leds_state); //点亮特定的灯
        leds_state += 1;
    }
    PROCESS_END();
}
```

6. **编译项目**：SMeshStudio 向导生成的代码可以直接使用，此时用户可以使用 Project——>Build Project 编译代码，如果没有错误的话，用户就可以在控制台看到如下内容，它表示用户程序的内存使用情况，其数值只要不超过 100%就可以了。

```
Building target: ContikiTest
Printing size:
"E:/SMeshStudio/smeshcore/tools/avr/bin/avr-size" --format=avr --
```

SMESH LINK

```
mcu=atmega128rfa1
"E:/SMeshStudio/workspace/ContikiTest/Release/ContikiTest.elf"
AVR Memory Usage
----------------
Device: atmega128rfa1

Program:   52648 bytes (40.2% Full)
(.text + .data + .bootloader)

Data:      12857 bytes (78.5% Full)
(.data + .bss + .noinit)
Finished building target: ContikiTest
18:07:51 Build Finished (took 27s.579ms)
```

7. **上载程序：**用户编译成功后就可以使用 mbed(Arduino_Contiki)——>upload sketch 上载程序，如果没有问题的话，用户可以看到下面的输出结果，至此，基于 SMeshStudio 的简单 Contiki 程序开发完毕：

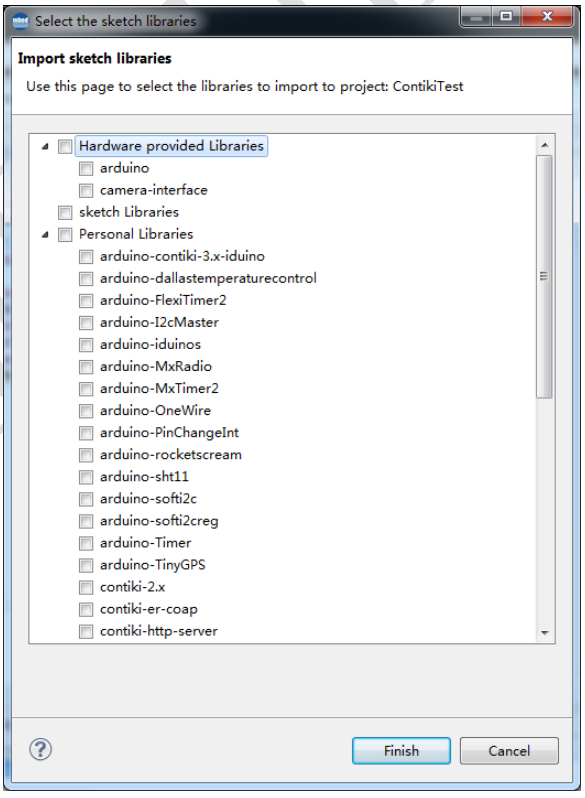
```
Starting upload
using mbed loader
Launching E:\SMeshStudio\smeshcore\tools\avr\bin\avrdude -
CE:\SMeshStudio\smeshcore\tools\avr\bin\avrdude.conf -patmega128rfa1 -
carduino -P COM7 -b57600 -
Uflash:w:E:\SMeshStudio\workspace\ContikiTest\Release\ContikiTest.hex:i
Output:
avrdude: AVR device initialized and ready to accept instructions
Reading | ################################################## | 100%
0.00s
avrdude: Device signature = 0x1ea701
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be
performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file
"E:\SMeshStudio\workspace\ContikiTest\Release\ContikiTest.hex"
avrdude: writing flash (52648 bytes):
Writing | ################################################## | 100%
12.56s
avrdude: 52648 bytes of flash written
avrdude: verifying flash memory against
E:\SMeshStudio\workspace\ContikiTest\Release\ContikiTest.hex:
avrdude: load data flash data from input file
E:\SMeshStudio\workspace\ContikiTest\Release\ContikiTest.hex:
avrdude: input file
E:\SMeshStudio\workspace\ContikiTest\Release\ContikiTest.hex contains
```

SMESH LINK

```
52648 bytes
avrdude: reading on-chip flash data:
Reading | ################################################## | 100%
9.78s
avrdude: verifying ...
avrdude: 52648 bytes of flash verified
avrdude done.  Thank you.

avrdude finished
upload done
```

3.2 SMeshStudio Contiki 扩展库应用

Contiki 官方提供了很多 app 应用程序，由于这些 app 在 Contiki 中就是一个 Process，所以用户可以把它们当成扩展库使用。SMeshStudio 提供的 Contiki 扩展库主要有两种类型，一种是必须在 Contiki 之上使用（在图中 Hardware provided Libraries 分类），另外一种是只要有 Contki 久可以使用（以后用户会知道 Contiki 也可以建立在 mbed 或 Arduino 的基础之上，在图中 Personal Libraries 目录下），当用户使用 mBed(Arduino_Contiki)——>Add a library to the selected project 菜单后就可以得到下面的界面：

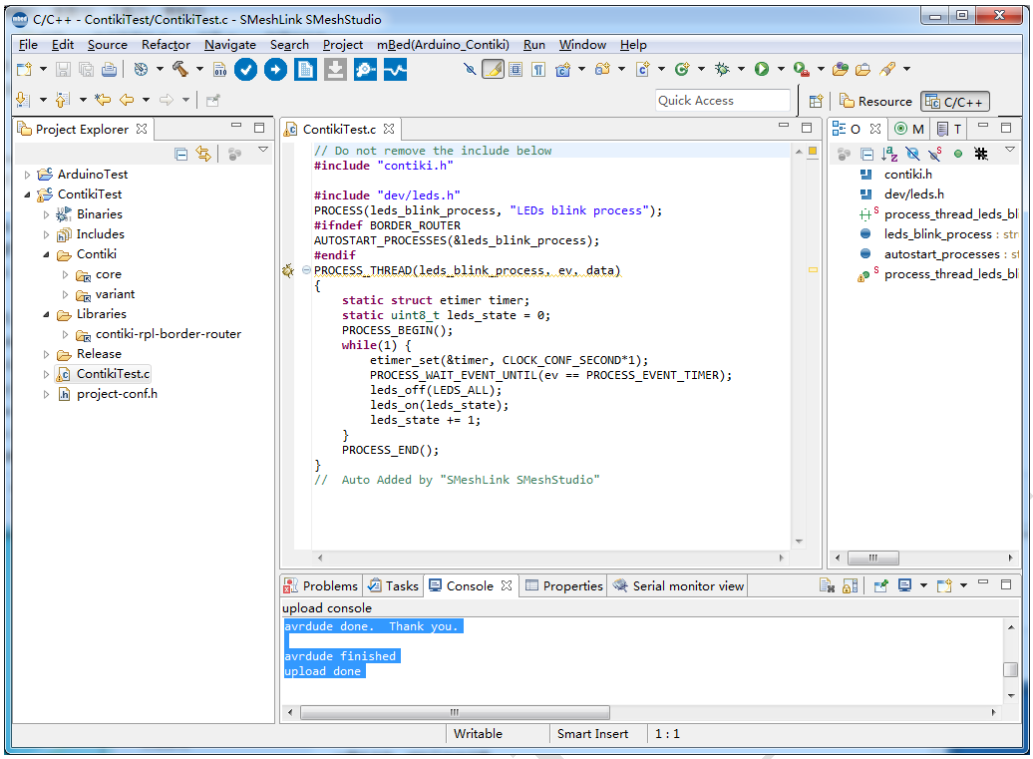


图表 3-6 SMeshStudio 导入 Contiki 扩展库

需要注意的是，由于 SMeshStudio 支持多个软件平台，所以它在 Personal Libraries 中也存放着多个平台的扩展库，用户在开发基于 Contiki 的应用时只能使用 contiki 开头的扩展库，接下来用户需要导入 contiki-rpl-border-router 库，该库用来实现 slip 协议，从而让计算机把

SMESHLINK

一个 Contiki 节点通过串口编成 IPV6 网卡，导入后，ContikiTest 项目的目录结构变化如下：



图表 3-7 SMeshStudio 导入扩展库目录变化

此时，用户会发现在导入的 rpl-border-router 库中有一个名为 project-conf-template.txt 的配置模板文件，用户用该文件中的内容去替换本项目的 project-conf.h 文件，该文件的内容是配合 border router 节点使用的，其中最关键的是以下几句，解释如下：

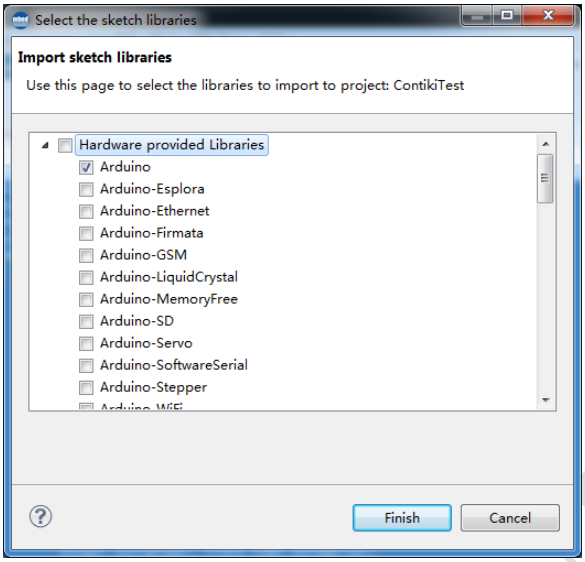
```
#define RDC_CONF_MCU_SLEEP 0
#define AVR_CONF_USE32KCRYSTAL 0   //border router节点一直运行在高功耗模式

#ifndef BORDER_ROUTER
#define BORDER_ROUTER //本节点是border router节点
#endif

#ifndef UIP_FALLBACK_INTERFACE
#define UIP_FALLBACK_INTERFACE rpl_interface
#endif
```

必要时可以修改 MAC 地址，6LowPan 网络中的所有节点 mac 地址不能相同。接下来重复前面的程序上载过程，就完成了 Contiki 扩展库的使用，当然，如果你要想要让此节点完成网络功能，你还必须学习更多的知识，用户可以从 mbed.smeshlink.com 寻找一些帮助。

3.3 SMeshStudio Contiki 使用 Arduino 扩展库

Contiki 虽然功能强大，但它在应用操作方面却远没有 Arduino 灵活，SMeshStudio 的强大之处在于你可以在 Contiki 的基础之上使用 Arduino，具体方法如下：

1. **导入面向 Contiki 的 Arduino 的扩展库**：面向 Contiki 的 Arduino 的扩展库存放在 Hardware provided Libraries 目录下，把 ArduinoTest 恢复到初始状态并导入 Arduino 扩展库和 Arduino-EEPROM 扩展库：



图表 3-8 SMeshStudio 导入面向 Contiki 的 Arduino 扩展库

2. **修改用户项目代码：**由于 Arduino 是面向 C++设计的 t，所以用户必须把 ContikiTest.c 也改成 C++文件，即把扩展名从.c 改成.cpp，然后在其中添加 Arduino 语法，该代码使用 Arduino 语法来控制灯，并使用 Arduino 语法读写 EEPROM，具体改动如下：

```
#include "Arduino.h"
#include "EEPROM/EEPROM.h"
PROCESS(leds_blink_process, "LEDs blink process");
AUTOSTART_PROCESSES(&leds_blink_process);
PROCESS_THREAD(leds_blink_process, ev, data)
{
    static struct etimer timer;
    static uint8_t value ,address,readval;
    PROCESS_BEGIN();
    pinMode(LED1,OUTPUT);
    pinMode(LED2,OUTPUT);
    for (address=0;address<10;address++)
    {
        EEPROM.write(address,value);
        readval=EEPROM.read(address);
        if (readval==value)
            digitalWrite(LED1,!digitalRead(LED1));
        else
            digitalWrite(LED2,!digitalRead(LED2));
        delay(100);
    }
    while(1) {
        etimer_set(&timer, CLOCK_CONF_SECOND*10);
        PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);

    }
    PROCESS_END();
}
```

程序上载后你会发现 LED1 会闪烁 5 次，这正是本程序正确执行的效果。

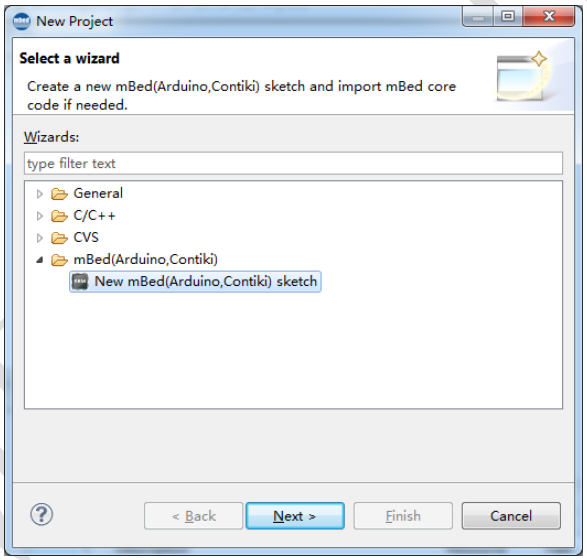
SMESHLINK

4 SMeshStudio mBed 入门指南

4.1 SMeshStudio mBed 快速入门

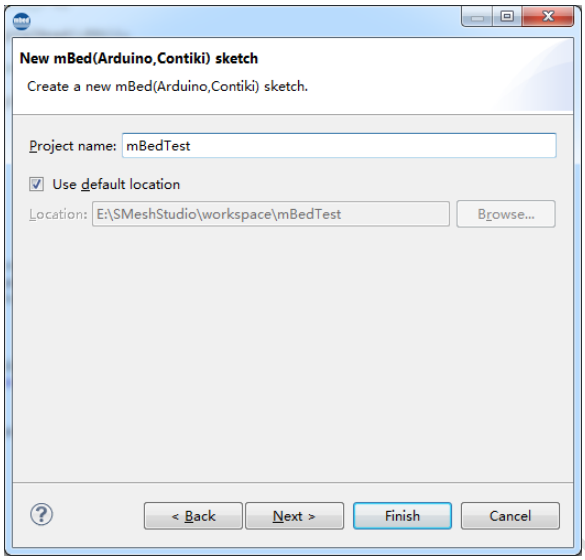
mBed 是基于 Arduino 理念面向 ARM CORTEX-M0/3/4 系列微处理器开发的快速原型开发平台，它具有极其广泛的硬件基础（它可以支持 ST，NXP，NORDIC，FREESCALE 等厂商的微处理器），同时，它还具备强大的硬件操控能力和网络连接能力，应该是未来最有前途的快速软件开发框架。但官方的 mBed 只支持在线开发，并不符合中国人的使用习惯，而且因为网速的原因还会经常导致在线开发系统崩溃，所以 SMeshStudio 借鉴了 Arduino 的开发思路，同样提供了完美的本地化开发平台，唯一的限制就是你的开发板必须支持 GCC 编译。其具体过程如下（其实大部分过程是一样的，但考虑到有些人只关心 Arduino，mBed 和 Contiki 中的一个，所以这里还是完整地重复了一遍）：

1. **选择 Eclipse 项目类型**：SMeshStudio 建立在标准 Eclipse 开发环境基础之上，它可以开发多种类型的应用程序，所以用户必须选择合适的开发类型，在 SMeshStudio 中，mBed, Arduino 和 Contiki 都属于同种类型。用户可以通过菜单 File——>New——>Projects 启动下面的项目类型选择界面并选择 New mBed(Arduino,Contiki) sketch 项目类型：



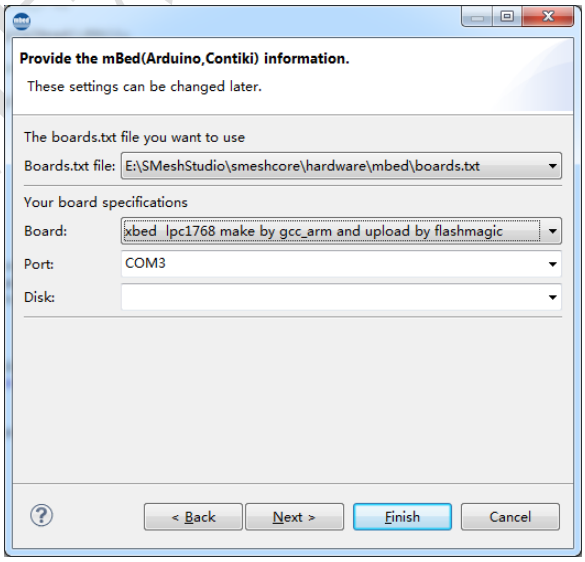
图表 4-1 SMeshStudio 项目类型选择

2. **设置项目名称**：用户选择 next 继续后出现项目名称设置界面，用户在这里可以随意设置喜欢的项目名称并设置项目保存位置（建议使用缺省者），用户在这里设置项目名称为 mBedTest：



图表 4-2 SMeshStudio 设置项目名称

3. **选择开发板类型**：用户选择 next 继续后出现开发板选择界面。SMeshStudio 支持多种软件开发平台，每个软件开发平台下又可以支持多个开发板，为了让用户的项目能匹配上用户的开发板，这一步的选择就显得非常重要。SMeshStudio 开发板的软件匹配原则如下：SMeshStudio 只根据开发板名称匹配用户项目，如果开发板名称中包含 bed，SMeshStudio 会把项目识别成 mBed 项目；如果包含 contiki，SMeshStudio 会把项目识别成 contiki 项目，否则 SMeshStudio 会把项目识别成 Arduino 项目。为了方便用户的选择，SMeshStudio 已经把不同的开发板放到不同的开发板描述文件中，所有用户在这里首先要选择开发板文件，然后再选择具体的开发板，之后用户还需要设置开发板上载程序使用的串口，考虑到有些开发板可以采用文件复制方式上载，SMeshStudio 也提供了磁盘选择选项，串口号和磁盘盘符必须设置一项后才能继续。用户在这里可以选择 xbed lpc1768，它是一块和官方 mbed lpc1768 兼容的 mBed 开发板，但添加了以太网接口，TF 卡接口，RF231 无线射频接口以及用户按键，从而使用户可以更好地应用 mBed 软件平台，是 SMeshStudio 强烈推荐的 mBed 开发板：

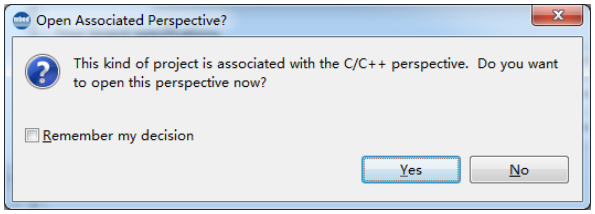


图表 4-3 SMeshStudio 选择开发板类型

4. **结束向导**：用户此时可以选择 Finish 结束向导，生成 mBed 项目，此时系统有可能提示说这是一个 C/C++项目，建议选择使用 C/C++视图，用户直接选择 yes 并建议选择

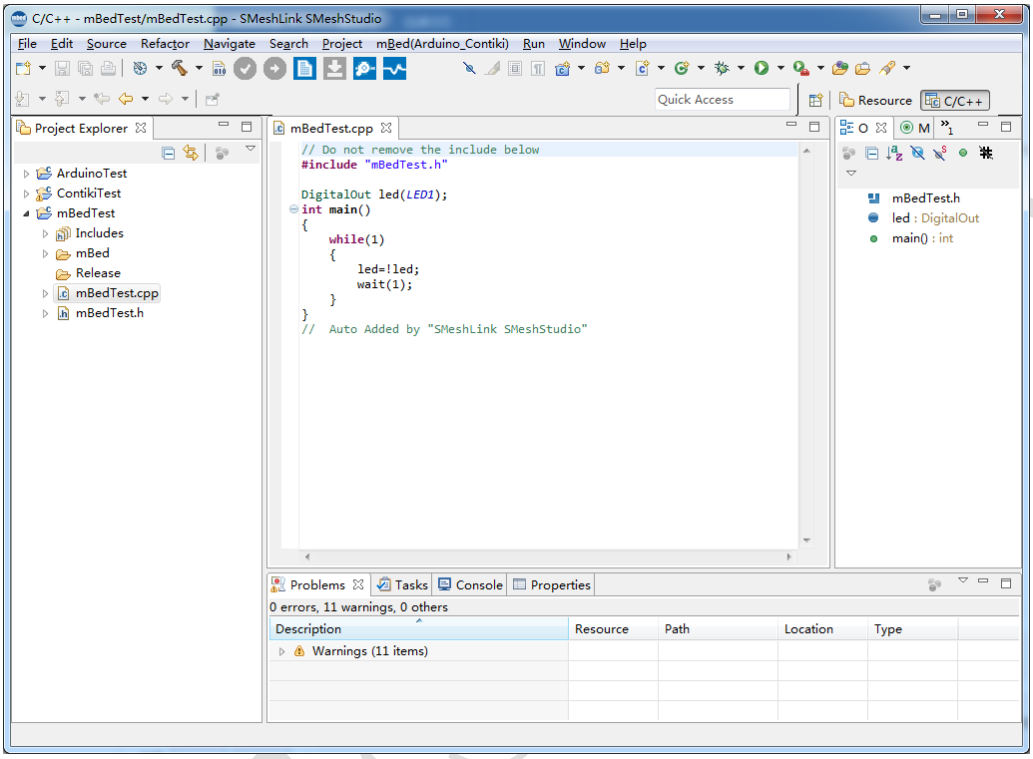
SMESHLINK

Remember my decision：



图表 4-4 SMeshStudio 视图类型选择

这样一来，用户就完成了 mBed 项目的创建过程，此时，SMeshStudio 界面如下：



图表 4-5 SMeshStudio 初始 mBed 项目

5. **编写代码**：此刻，用户可以来简单地理解一下 mBedTest 项目，任何基于 SMeshStudio 的 mBed 项目的代码都由三部分组成，具体列表如下：

- mBed 核心库，即 Project Explorer 视图中的 Contiki 目录，该目录下有两个文件夹，一个是 core，里面放的是 mBed 独立于硬件部分的实现；另外一个是 variant，里面放的是 mBed 和硬件相关的实现，其中一个是 cmsis，里面放的是和系统启动并和编译其相关的代码，另外一个是 hal，里面放的是和具体的开发板相关的硬件抽象实现代码。
- Libraries 扩展库，即 Project Explorer 视图中的 Libraries 目录，里面存放着用户导入的和特定应用相关的扩展库（由于本项目没有导入扩展库，所以本项目中并不存在该目录，但在接下来的例子中你会看到）。
- 用户项目代码，向导默认会生成两个文件，一个是和项目同名的 cpp 文件，用于完成项目项目，另外一个则是和项目同名的.h 文件，它的内容就是包含了 mbed.h 文件。

接下来用户把 mBedTest.cpp 的内容改动如下（原先向导生成的代码只是 1 秒钟变换一次灯，改动后增加了输出 Hello World！功能）：

```
#include "mBedTest.h"
```

```
static int count=0;
DigitalOut led(LED1);
int main()
{
    while(1)
    {
        led=!led;
        printf("Hello World, count id =%d.\n!",count++);
        wait(1);
    }
}
```

6. **编译项目**：代码编写完毕后，用户可以使用 Project——>Build Project 编译代码，如果没有错误的话，用户就可以在控制台看到如下内容，它表示用户程序的内存使用情况，用户只要在 Eclipse 的 Console 中没有看到错误即可。

```
Building target: mBedTest
Printing size:
"E:/SMeshStudio/smeshcore/tools/gcc_arm/bin/arm-none-eabi-size" -A
"E:/SMeshStudio/workspace/mBedTest/Release/mBedTest.elf"
E:/SMeshStudio/workspace/mBedTest/Release/mBedTest.elf  :
section            size        addr
.text             25848           0
.ARM.exidx            8       25848
.data               224   268435656
.bss                828   268435880
.heap              2048   268436712
.stack_dummy       3072   268436712
.ARM.attributes      41           0
.comment            112           0
.debug_frame       3824           0
.stabstr            118           0
Total             36123
```

7. **上载程序**：用户编译成功后就可以使用 mbed(Arduino_Contiki)——>upload sketch 上载程序，如果没有问题的话，用户可以看到下面的输出结果，至此，基于 SMeshStudio 的简单 mBed 程序开发完毕（我在测试中发现，USB3.0 会有问题，所以建议用户使用 USB2.0 端口）：

```
Starting upload
using mbed loader
Launching E:\SMeshStudio\smeshcore\tools\utils\fm COM(3, 115200)
HARDWARE(BOOTEXECRTS, 50, 100) DEVICE(LPC1768, 0.000000, 0)
ERASE(DEVICE, PROTECTISP)
HEXFILE(E:\SMeshStudio\workspace\mBedTest\Release\mBedTest.hex,
NOCHECKSUMS, NOFILL, PROTECTISP)
VERIFY(E:\SMeshStudio\workspace\mBedTest\Release\mBedTest.hex,
NOCHECKSUMS)
```

SMESHLINK

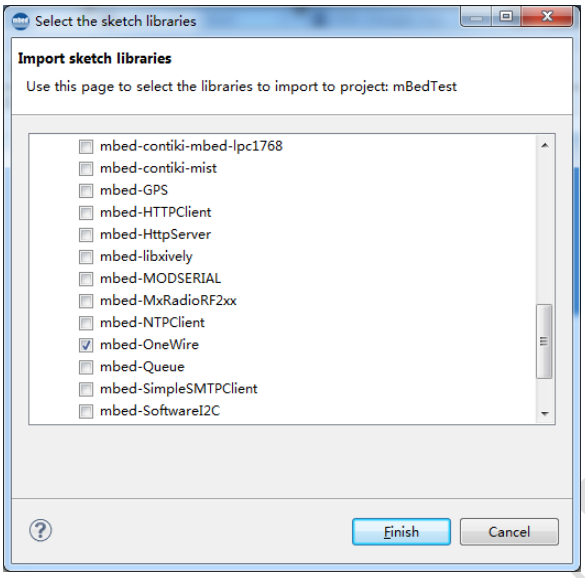
```
Output:
Flash Magic Version 7.66.3343
8051/XA Driver Version 3.04.3296
ARM UART Driver Version 3.11.3302
ARM Cortex UART Driver Version 4.63.3312
ARM Ethernet Driver Version 2.05.3296
ARM Cortex Ethernet Driver Version 2.07.3296
ARM CAN Driver Version 2.11.3296
(C) Embedded Systems Academy 2000-2013 All rights reserved
NON PRODUCTION USE ONLY
Connected
Device selected
Erase complete (DEVICE)
Hex file programming complete
(E:\SMeshStudio\workspace\mBedTest\Release\mBedTest.hex)
Verify passed
(E:\SMeshStudio\workspace\mBedTest\Release\mBedTest.hex)
fm finished
upload done
```

此时你就可以通过类似于AccessPort之类的串口调试工具来查看mBed的输出了，mBed默认的波特率是9600。

4.2 SMeshStudio mBed 扩展库应用

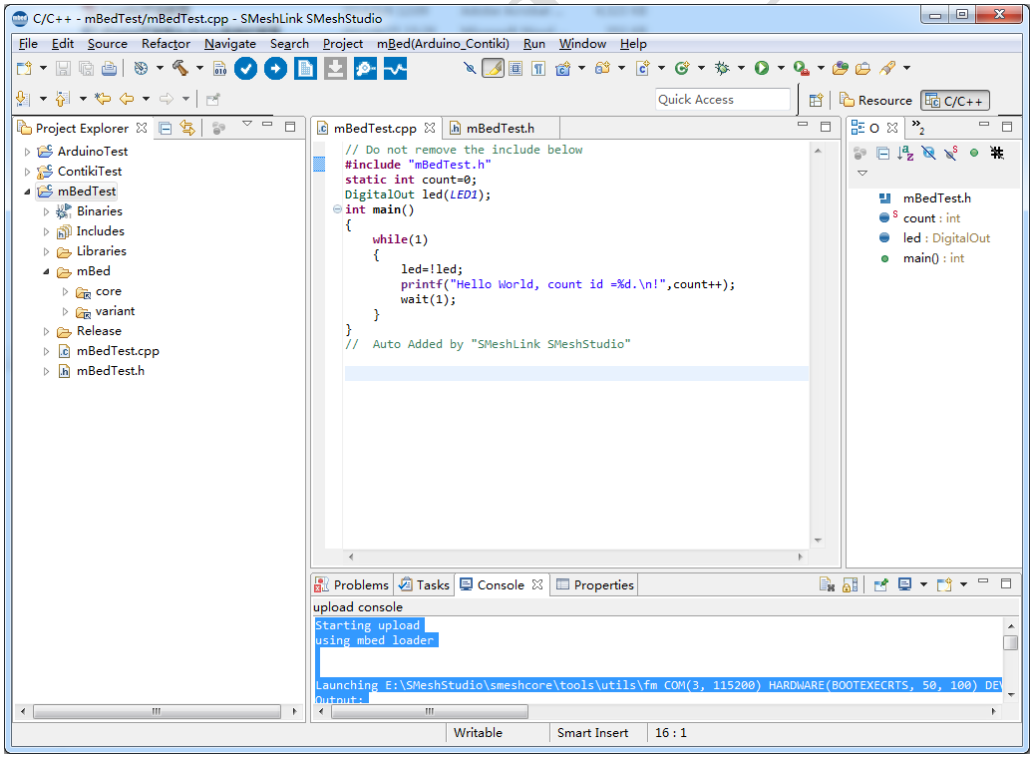
mBed 官方和第三方提供了大量的扩展库，它们的使用可以大大简化用户应用的开发，这也是 mBed 的强大之处。SMeshStudio 提供的 mBed 扩展库主要有两个部分，一个是 mBed 官方提供的，另外一个是第三方提供的，当用户使用 mBed(Arduino_Contiki)——>Add a library to the selected project 菜单后就可以得到下面的界面。mBed 官方提供的官方库在 Hardware provided Libraries 分类下，而第三方提供的包括 SMeshStudio 自带的则在 Personal Libraries 目录下：

SMESHLINK



图表 4-6 SMeshStudio 导入 Contiki 扩展库

需要注意的是，由于 SMeshStudio 支持多个软件平台，所以它在 Personal Libraries 中也存放着多个平台的扩展库，用户在开发基于 mBed 的应用时只能使用 mBed 开头的扩展库，接下来用户可以导入 mbed-OneWire 库，该库实现了单总线协议，从而让用户可以很方便地读去 xbed lpc1768 上的单总线设备 ds2411，导入后，mBedTest 项目的目录结构变化如下：



图表 4-7 SMeshStudio 导入扩展库目录变化

接下来，用户需要把 mBedTest 内容修改如下，它将在系统启动时打印 ds2411 的值：

```
#include "mBedTest.h"
#include "OneWire.h"
static int count=0;
DigitalOut led(LED1);
OneWire ow(P1_29);
```

```
int main()
{
    char romcode[8];                    // Array for ROM-Code
    ow.busInit();
    ow.getRomCode(romcode);          // Get ROM-Code
    printf("Ds2411 rom is %X.%X.%X.%X.%X.%X.%X.%X
\n",romcode[0],romcode[1],romcode[2],romcode[3],romcode[4],romcode[5],romcode[6],ro
mcode[7]);

    while(1)
    {
        led=!led;
        printf("Hello World, count id =%d.\n!",count++);
        wait(1);
    }
}
```

上载成功后，你就可以通过类似于 AccessPort 之类的串口调试工具来查看 mBed 的输出了（默认波特率是 9600），你可以得到下面的输出：

```
Ds2411 rom is 1.CD.6E.C2.13.0.0.CB
Hello World, count id =0.
!Hello World, count id =1.
!Hello World, count id =2.
!Hello World, count id =3.
```

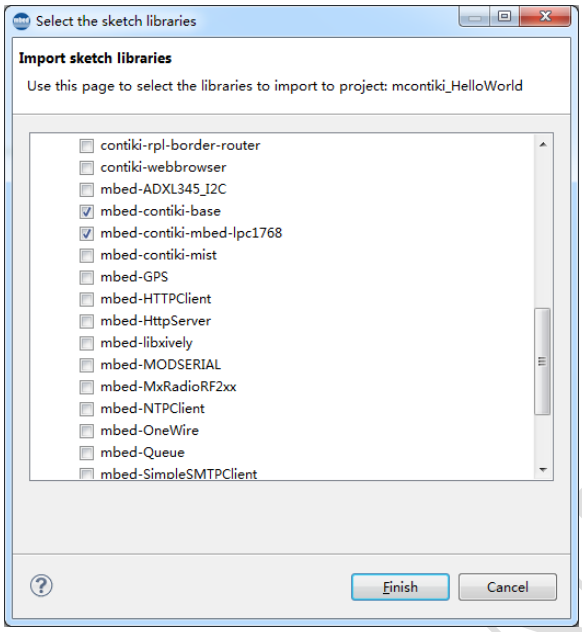
这样就完成了 mBed 扩展库的使用，如果你要想要让此节点完成更复杂的功能，你还必须学习更多的知识，用户可以从 mbed.smeshlink.com 寻找一些帮助。

4.3 SMeshStudio mBed 使用 Contiki 扩展库

mBed 虽然使用方便，但它却没有无线功能，也没有 IPV6 功能，而这些都是 Contiki 的强项，SMeshStudio 的强大之处在于你可以在 mBed 的基础之上使用 Contiki，具体方法如下：

1. **导入面向 mBed 的 Contiki 扩展库**：面向 mBed 的 Contiki 扩展库存放在 Personal Libraries 目录下，一共有两个，一个是和硬件无关的实现 mbed-contiki-base，另外一个是和板卡相关的实现，这里用的是 mbed-contiki-mbed-lpc1768：

SMESHLINK



图表 4-8 SMeshStudio 导入面向 mBed 的 Contiki 扩展库

2. 修改用户项目代码：首先需要添加项目配置文件，用户需要把 mbed-contiki-mbed-lpc1768 目录下的 project-conf.h.template 文件复制到项目目录下，并改名为 project-conf.h，该文件完成 contiki 必备的宏定义工作，其主要内容解释如下：

```
#ifndef _t_H_
#define _t_H_
//去掉编译器自动添加的ARDUINO宏定义
#undef ARDUINO
//设置节点的mac地址定义方式0表示使用ds2411自动设置，1表示使用下面的
EUI64_ADDRESS手动设置
#define UIP_CONF_EUI64    1
//设置节点的功耗模式，当前系统还不支持低功耗，所以这里只能设成1
#define LOWPOWER 0
//设置节点的mac地址，该定义只有在UIP_CONF_EUI64为1时有效
#define EUI64_ADDRESS {0x02, 0, 0, 0, 0, 0, 0, 0xbb};
/* 设置节点的工作频段，范围是11-26*/
#define RFCHANNEL 26
//下面是在系统编译时会自动添加的宏定义
//#define AUTOSTART_ENABLE 1
//#define UIP_CONF_IPV6 1
//#define RF230BB 1
//#define WATCHDOG_CONF_TIMEOUT WDTO_8S
//#define AUTO_CRC_PADDING 2
//#define HAVE_STDINT_H
#endif /* _t_H_ */
```

然后，修改 mBedTest.cpp 内容如下（Contiki 不需要 main 函数）：

```
#include "mBedTest.h"
#include "OneWire.h"
DigitalOut led(P2_6);
OneWire ow(P1_29);
#ifdef __cplusplus
extern "C" { //C++调用C必须的
```

SMESHLINK

```
#endif
#include "contiki.h"
#ifdef __cplusplus
}
#endif
PROCESS(leds_blink_process, "LEDs blink process");
AUTOSTART_PROCESSES(&leds_blink_process);

PROCESS_THREAD(leds_blink_process, ev, data)
{
    static struct etimer timer;
    static char romcode[8];                    // Array for ROM-Code
    PROCESS_BEGIN();
    ow.busInit();
    ow.getRomCode(romcode);          // Get ROM-Code
    printf("Ds2411 rom is %X.%X.%X.%X.%X.%X.%X.%X
\n",romcode[0],romcode[1],romcode[2],romcode[3],romcode[4],romcode[5],romcode[6],ro
mcode[7]);
    while(1) {
        etimer_set(&timer, CLOCK_CONF_SECOND*1);
        PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);
        led=!led;
    }
    PROCESS_END();
}
```

程序上载后你会发现 LED1 会 1 秒闪烁 1 次，并同样也会输出 DS2411 的值，这正是本程序正确执行的效果。

当 mBed 导入 Contiki 扩展库后，也就可以使用 Contiki 的扩展库了，在这里用户可以导入 contiki-rpl-border-router 用以实现 SLIP 协议。导入后，用户可以发现在导入的 rpl-border-router 库中有一个名为 project-conf-template.txt 的配置模板文件，用户需要用该文件中的内容去替换本项目的 project-conf.h 文件，该文件的内容是配合 border router 节点使用的，其中最关键的是以下几句，解释如下：

```
#define RDC_CONF_MCU_SLEEP 0
#define AVR_CONF_USE32KCRYSTAL 0   //border router节点一直运行在高功耗模式

#ifndef BORDER_ROUTER
#define BORDER_ROUTER //本节点是border router节点
#endif

#ifndef UIP_FALLBACK_INTERFACE
#define UIP_FALLBACK_INTERFACE rpl_interface
#endif
```

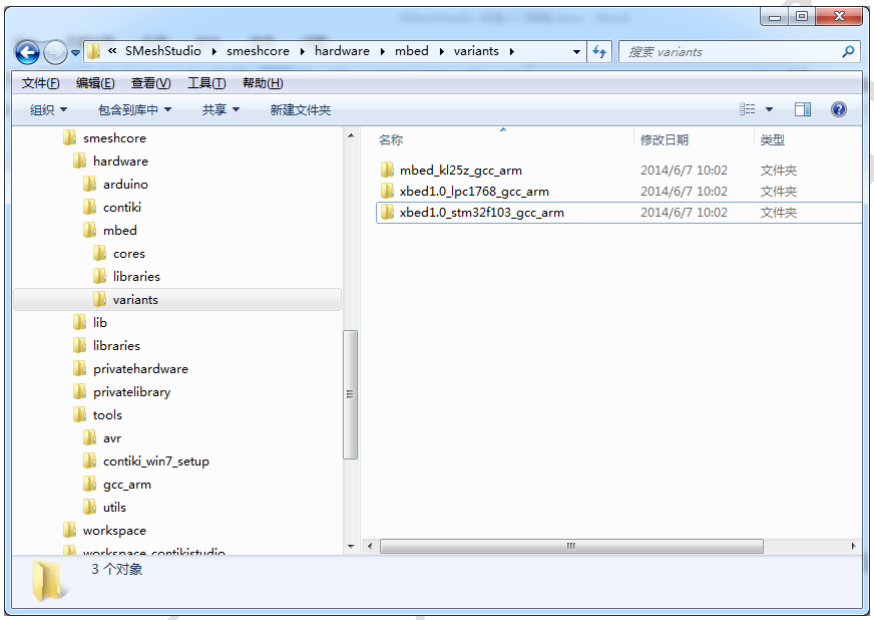
必要时可以修改 MAC 地址，6LowPan 网络中的所有节点 mac 地址不能相同。接下来用户必须删除 mBedTest.cpp 文件，因为 contiki-rpl-border-router 本质上就是一个应用程序，不删除会发生冲突，然后重复前面的编译与程序上载过程，从而完成 mBed 之上的 Contiki 扩展库的使用，当然，如果你要想要让此节点完成网络功能，你还必须学习更多的知识，用户可以从 mbed.smeshlink.com 寻找一些帮助。

SMESHLINK

5 SMeshStudio 实现机制

SMeshStudio 是一个开放平台，理论上来说，任何支持 GCC 编译的软件都可以通过它来开发，当然， SMeshStudio 默认支持的开发板是有限的，尤其是 mBed 开发板，SMeshStudio 支持的只是众多 mBed 开发板中的几个，那么用户能否自己添加开发板呢？答案是肯定的，不过你要注意 SMeshStudio 的授权方式，它只授权在 SMeshLink 生产或销售的开发板上使用，当然，如果你只是自己用，并不打算复制给它认，这应该还是可以的。

下图是 SMeshStudio 的完整目录结构，其中的 eclipse/eclipse_x64 就是安装了 SMeshStudio 插件的 eclipse 系统，前者是 32 位版本，后者是 64 位版本，下面的 smeshcore 就是各类软件的核心库及编译工具，这是用户需要关心的主要内容。



图表 5-1 SMeshStudio **目录结构**

下面是 smeshcore 目录内容的完整介绍：

- **Hardware:**其下有 arduino,contiki,mbed 三个目录，里面存放的就是各自的 SDK 代码库，每个目录下又会有 cores,libraries 和 variants 三个目录，分别存在着各自 SDK 库的硬件无关实现部分，扩展库部分和硬件相关实现部分。Hardware 目录下还存放着名为 platform.txt 和 boards.txt 两个文本文件，分别用于存放编译本平台的配置和编译本平台下特定板卡的配置。
- **libraries:**里面存放着官方提供的能在所有平台下使用的扩展库，一般都是空的。
- **privatelibrary:**里面存放着 SMeshStudio 和第三方提供的扩展库，具体对应于哪个平台要看扩展库的前缀，如 mbed 开头的就是 mbed 的扩展库。
- **tools：**里面存放的是各个平台编译上载需要的工具软件，当前支持 avr-gcc 编译和 arm-gcc 编译，并支持 fm,avrdude,dfu 三种程序上载方式。

SMeshStudio 工作时会自动读取相应平台的 platform.txt 和 boards.txt 文件，从而确定从那载入相应的库函数，所以用户添加自定义开发板的关键就是修改这两个文件，实际上，对于 mbed 来说，使用的编译器是一样的，所以只要修改 boards.txt 文件即可。下面，以添加 mbed stm32f103 开发板为描述一下整个过程，具体步骤如下：

1. **准备目录结构：**mbed 官方已经支持 stm32f103 微处理器，这样用户就有可能把 stm32f103 相关的开发板添加到 SMeshStudio 开发环境支持的板卡中来，用户从

https://github.com/mbedmicro/mbed 下载 mbed 的完整库，解压后用户只需要关心其中的 /libraries / mbed / targets 目录即可，用户需要从中得到 mbed stm32f103 硬件相关的实现。用户首先在 variants 目录下创建 mbed-NUCLEO_F103RB 目录，用以保存它相应的实现文件，并在之下创建 cmsis 和 hal 两个目录，这和 mbed 的实现是相对应的。

2. **准备目录内容**：接下来需要准备面向 SMeshStudio 的目录结构，用户需要复制下载的\libraries\mbed\targets\cmsis\TARGET_STM\TARGET_NUCLEO_F103RB 目录的所有内容包括目录到上面的 cmsis 目录下，并删除其中没有用的 TOOLCHAIN_ARM_MICRO 和 TOOLCHAIN_ARM_STD 两个目录，另外一定要注意的是由于 Eclipse 不支持.s 扩展名的文件，用户需要把 startup_stm32f10x.s 文件改成 startup_stm32f10x.S 文件，即改成大小；另外复制\libraries\mbed\targets\hal\TARGET_STM\TARGET_NUCLEO_F103RB 目录的所有内容包括目录到上面的 hal 目录下，这样就完成了目录内容的准备。

3. 修改 boards.txt 文件：当用户添加了开发板以后必须在 boards.txt 中添加相应内容，用户在这里需要添加的内容如下，具体内容的解读已经超出了本书的内容，用户可以查看相关的 GCC 使用手册：

```
mbed_stm32f103_gcc_arm.name=MBED NUCLEO_F103RB make by gcc_arm
mbed_stm32f103_gcc_arm.upload.protocol=dfu
mbed_stm32f103_gcc_arm.upload.tool=mbed_dfu
mbed_stm32f103_gcc_arm.build.mcu=cortex-m3
mbed_stm32f103_gcc_arm.build.cpu=STM32F103RB

mbed_stm32f103_gcc_arm.build.architecture=arm-none-eabi
mbed_stm32f103_gcc_arm.build.command.as=arm-none-eabi-as
mbed_stm32f103_gcc_arm.build.command.gcc=arm-none-eabi-gcc
mbed_stm32f103_gcc_arm.build.command.g++=arm-none-eabi-g++
mbed_stm32f103_gcc_arm.build.command.ar=arm-none-eabi-ar
mbed_stm32f103_gcc_arm.build.command.objcopy=arm-none-eabi-objcopy
mbed_stm32f103_gcc_arm.build.command.objdump=arm-none-eabi-objdump
mbed_stm32f103_gcc_arm.build.command.size=arm-none-eabi-size

mbed_stm32f103_gcc_arm.build.core=mbed
mbed_stm32f103_gcc_arm.build.variant= mbed-NUCLEO_F103RB

mbed_stm32f103_gcc_arm.build.CC_FLAGS= -mcpu=cortex-m3 -mthumb -c -Os -fno-
common -fmessage-length=0 -Wall -fno-exceptions -ffunction-sections -fdata-
sections
mbed_stm32f103_gcc_arm.build.ONLY_C_FLAGS= -std=gnu99
mbed_stm32f103_gcc_arm.build.ONLY_CPP_FLAGS= -std=gnu++98
mbed_stm32f103_gcc_arm.build.CC_SYMBOLS = -DTARGET_STM -DTARGET_M3  -
DSTM32F10X_MD  -DTOOLCHAIN_GCC_ARM -DTOOLCHAIN_GCC -D__CORTEX_M3 -DARM_MATH_CM3
-DSTM32F1 -DSTM32F103RB -DSTM
mbed_stm32f103_gcc_arm.build.LD_FLAGS =  -mcpu=cortex-m3 -mthumb -Wl,--gc-
sections --specs=nano.specs -u _printf_float -u _scanf_float
mbed_stm32f103_gcc_arm.build.LD_SYS_LIBS =  -lstdc++ -lsupc++ -lm -lc -lgcc -
lnosys
```

SMESH LINK

```
mbed_stm32f103_gcc_arm.build.ldscript=hardware/mbed/variants/ mbed-
NUCLEO_F103RB/cmsis/TARGET_STM/TARGET_NUCLEO_F103RB/TOOLCHAIN_GCC_ARM/STM32F10X
.ld

mbed_stm32f103_gcc_arm.build.use_archiver=false
mbed_stm32f103_gcc_arm.recipe.S.o.pattern="{compiler.gcc_arm.path}arm-none-
eabi-gcc" {build.CC_FLAGS} {build.CC_SYMBOLS} {build.ONLY_C_FLAGS}   {includes}
"{source_file}" -o "{object_file}"
mbed_stm32f103_gcc_arm.recipe.c.o.pattern="{compiler.gcc_arm.path}arm-none-
eabi-gcc" {build.CC_FLAGS} {build.CC_SYMBOLS} {build.ONLY_C_FLAGS}   {includes}
"{source_file}" -o "{object_file}"
mbed_stm32f103_gcc_arm.recipe.cpp.o.pattern="{compiler.gcc_arm.path}arm-none-
eabi-g++" {build.CC_FLAGS} {build.CC_SYMBOLS} {build.ONLY_CPP_FLAGS} {includes}
"{source_file}" -o "{object_file}"
mbed_stm32f103_gcc_arm.recipe.ar.pattern="{compiler.gcc_arm.path}arm-none-eabi-
ar" -r "{build.path}/{archive_file}" {object_file}
mbed_stm32f103_gcc_arm.recipe.c.combine.pattern="{compiler.gcc_arm.path}arm-
none-eabi-gcc" {build.LD_FLAGS}  "-T{runtime.ide.path}/{build.ldscript}" -o
"{build.path}/{build.project_name}.elf" {object_files} -L"{build.path}" -
L"{runtime.ide.path}/hardware/mbed/variants/ mbed-NUCLEO_F103RB
/cmsis/TARGET_STM/TARGET_NUCLEO_F103RB/TOOLCHAIN_GCC_ARM" {build.LD_SYS_LIBS}
mbed_stm32f103_gcc_arm.recipe.objcopy.bin.pattern="{compiler.gcc_arm.path}arm-
none-eabi-objcopy"  -Obinary "{build.path}/{build.project_name}.elf"
"{build.path}/{build.project_name}.bin"
mbed_stm32f103_gcc_arm.recipe.objcopy.hex.pattern="{compiler.gcc_arm.path}arm-
none-eabi-objcopy"  -Oihex "{build.path}/{build.project_name}.elf"
"{build.path}/{build.project_name}.hex"
mbed_stm32f103_gcc_arm.recipe.size.pattern="{compiler.gcc_arm.path}arm-none-
eabi-size" -A "{build.path}/{build.project_name}.elf"
```

这里的 mbed_dfu 在 platform.txt 中定义，它确定了文件上载的具体方式，具体定义如下（但实际上这块板子并不支持 dfu 方式上载，这里只是举了个例子）：

```
tools.mbed_dfu.upload.pattern="{runtime.ide.path}/tools/utils/dfu-util" -a1 -d
0x1EAF:0x0003 -D  {build.path}/{build.project_name}.bin -R
```

SMESHLINK