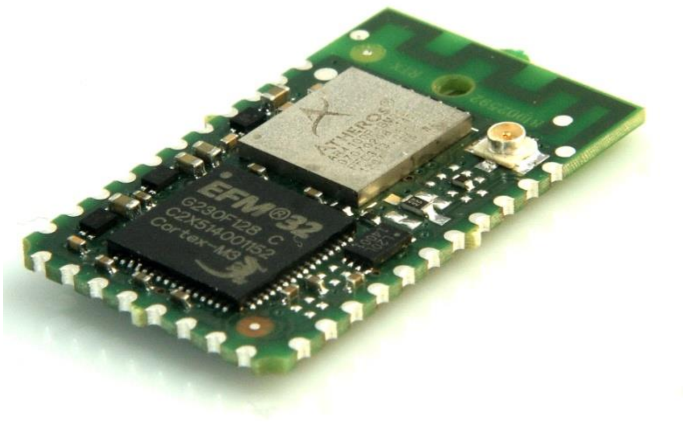
RTX41xx

Wi-Fi Modules

Variants covered in this document:
RTX4100
RTX4140



User Guide UG3

Application Development

CONTENT

1 Introduction

1.1 General Description

The RTX41xx (RTX4100 or RTX4140) Wi-Fi Module is a small form-factor, single stream, 802.11b/g/n Wi-Fi module with on-board low power application processor. It is targeted at applications that send infrequent data packets over the network. Typically, 802.11 applications addressed by a RTX41xx module will place a priority on low power consumption, ease of development, and system integration.

This document serves as a quick start guide for simple application building. This guide is not intended to provide a complete description of application development for the RTX41xx, but to provide an overview of steps involved.

The information in this document only needed for building custom collocated applications. The reader is expected to have experience with embedded software.

The application development process can be divided into three steps:

1. Installing the necessary tools, see tools installation guide ([UG2])
2. **Developing, building and downloading the application**
3. Debugging the application, see application debugging guide ([UG4])

This document covers step 2 in the process above. For steps 1 and 3, refer to other user guides.

NOTE: The instructions in this guide should be completed after completing step 1 and before step 3 above.

1.2 Module variants covered by this document

This document covers both the RTX4100 and RTX4140 WiFi modules. When RTX41xx is mentioned, in this document, it will be referring to ***both*** RTX4100 and RTX4140.

1.3 Document History

V1.0 Official release	MAD	2012-07-11
V1.1 Added Applications Flow Charts	MAD	2012-08-28
V1.2 Added Application Flow Chart (SoftAP TcpServer)	MAD	2012-10-01
V1.3 Added TempSensor SoftAP and Web Server Applications.	MAD	2012-11-01
V1.4 Added description of the HW re-sources used by the Amelie Platform FW. Restructured the document.	LKA	2012-11-05
V1.5 Added Section 5, cloud applications	MAD	2012-12-07
V1.6 Removed EVK references	TM	2013-02-19
V1.7 Added sections on Application frame-work, RTX OS and application development	MH	2013-03-19
V1.8 Added RTX4140, changed description of COM port selection for CoLA controller	TM	2013-06-04
V1.9 Added description on new tool chain from SDK version 1.6.x.x and up	PM	2013-10-23
V1.10 Updated the description of the Tcp2Uart application with description of software flow control etc.	LKA	2013-12-06

Disclaimer: This document can be subject to change without prior notice.

1.4 SW/HW Version

This document is applicable for the following versions.

- SDK Version 1.6.0.x (or newer)
- Platform Version 1.6.0.x (or newer)
- WSAB version V3RA (or newer)

1.5 Document References

[DS1]. RTX4100_Datasheet_DS1.pdf.
[DS2]. RTX4140_Datasheet_DS2.pdf.
[IS1]. AmelieApi_Vxxxx.pdf.
[QS6]. RTX4100_Quick_Start_Guide_Exosite_QS6.pdf.
[QS7]. RTX4100_Quick_Start_Guide_Nabto_QS7.pdf.
[QS8]. RTX4100_Quick_Start_Guide_2Lemetry_QS8.pdf.
[QS9]. RTX4100_Quick_Start_Guide_Sensinode_QS9.pdf.
[UG1]. RTX4100_User_Guide_Module_Evaluation_UG1.pdf.
[UG2]. RTX4100_User_Guide_Tools_Installation_UG2.pdf.
[UG3]. RTX4100_User_Guide_Application_Development_UG3.pdf.
[UG4]. RTX4100_User_Guide_Application_Debugging_UG4.pdf.

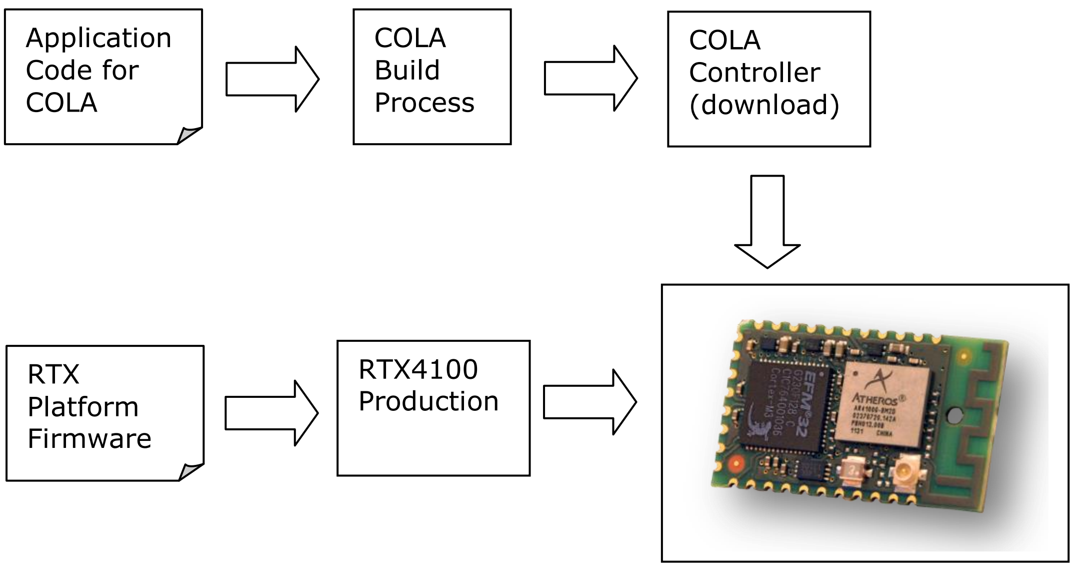
2 The SDK Content

This section provides a brief overview of the RTX41xx Software Development Kit. The name “Amelie” denotes the software development kit for a range of RTX’s low power Wi-Fi modules, and will be used throughout the documentation.

AmelieSdk_vX.X.X.X.exe performs an installation. Refer to ([UG2]) for installing the SDK.

CoLA is an abbreviation for Co-Located Application, a user application that resides next to the Platform firmware. When the CoLA is loaded into the flash at its proper location, the Platform firmware detects the presence of this CoLA and executes it as a single task under its Operating System. The CoLA concept makes it easy for application programmers to build applications while still having access to all the features of the underlying system firmware through Application Programmers Interfaces (APIs). For a complete overview of the API see reference ([IS1]).

The application development steps are illustrated below:



The RTX41xx comes pre-loaded with RTX platform firmware, with support for Co-Located Applications (CoLA). The application developer can then build their own applications using the APIs defined by the platform firmware, and download the application into the module for execution.

The pre-loaded CoLA in the Wireless Sensor Application Board (WSAB) supplied with the DVK is the Terminal reference application, please refer to the user guide for module evaluation ([UG1])

Refer to the User Guide Module Evaluation ([UG1]) for information on the terminal reference application commands. If you are new to the RTX4100 SDK, we suggest you start to follow the instructions in the Module evaluation User Guide and familiarize yourself with the RTX4100. Then, you can look into the Terminal application source code to see the underlying implementation.

The SDK delivery from RTX includes the following items:

- Sample code implementing the following applications:
  - Blinky: Small example application that uses a timer to flash a LED on the demo board.
  - TCP Server/client
  - UDP Server/client
  - Terminal Application
  - 3 different Temperature Sensor implementations, demonstrating various low power capabilities.
  - TempSensorSoftApCfg, demonstrating how it is possible to configure a module by booting it in SoftAP mode with telnet support.
  - SoftApTcpServer, demonstrating an AP implementation with TCP server.
  - WebServer, demonstrating a very simple WEB server implementation.
  - Tcp2Uart, demonstrates a simple implementation of a data pipe
  - Cloud reference applications, demonstrating various cloud solutions
- Header files for available APIs.
- Build environment: Build scripts, linker files etc. are provided for ARM GCC Embedded tool chain.
- RTX EAI Port Server: PC application used for communicating with the hardware.
- CoLA Controller DLL: Used for selecting active image and updating firmware.
- CoLA Controller GUI: Example application using the DLL. In the end product the DLL could be used directly from e.g. an Eclipse IDE.

3 Software Architecture

3.1 Platform Overview

The RTX41xx Wi-Fi Module contains a Qualcomm Atheros AR4100P Wi-Fi SIP chip and an Energy Micro application Micro Controller Unit (MCU). The application processor has internal Flash and RAM. The Wi-Fi module boots from a dedicated serial Flash. The processor is powered by an LDO with low power consumption to keep the total standby current very low. Furthermore, the application processor controls two additional LDO's to power the Wi-Fi module and the serial Flash. The Wi-Fi AR4100P chip can be turned off to enter an ultra-low power mode.

A number of I/O's are available to allow a wide range of applications. These include timers, serial communication interfaces, analog comparators, Analog-to-Digital Converters, Digital-to-Analog Converters, crystal oscillators, and a debug interface. Please see the RTX4100 and RTX4140 data sheets for details.

3.2 SW Components

The RTX41xx contains two major components; the Wi-Fi module and the Application MCU. The Application MCU contains all the necessary software components to implement a complete Wi-Fi device, including the application.

The diagram in Figure 1 conceptually illustrates the software architecture of the application MCU. The software architecture is split in the following two parts:

1. The user application – Also known as the Co-Located Application (CoLA).
2. The Module Firmware, also called Platform Firmware

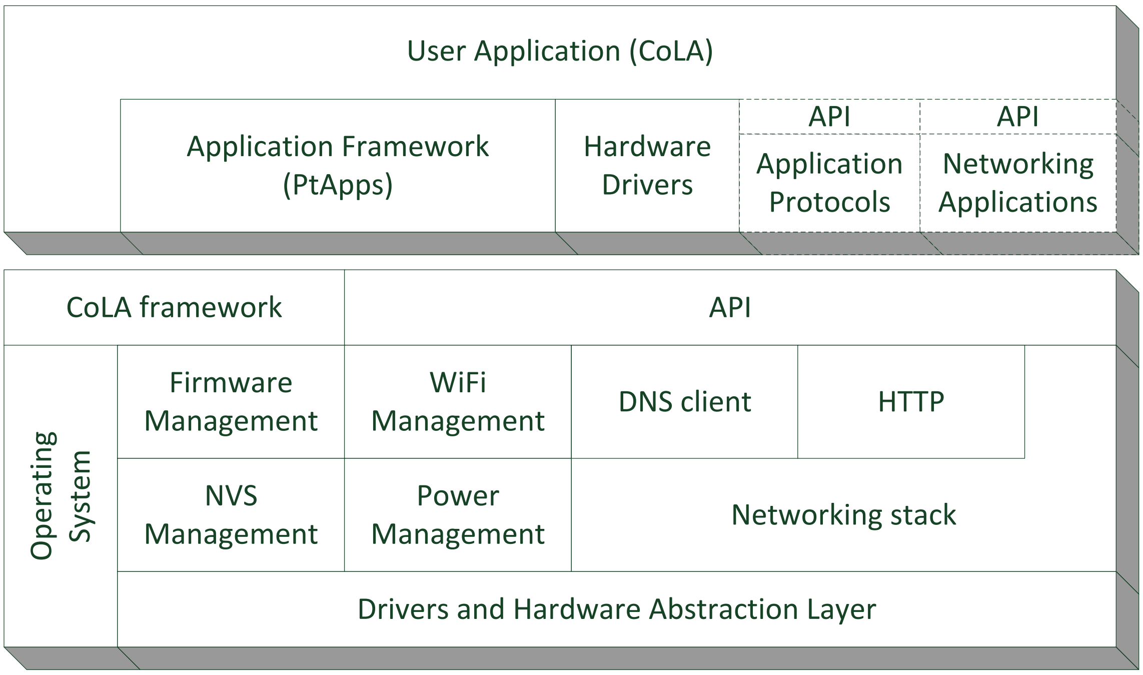


Figure 1: Overview of the SW Architecture on the Application MCU

The RTX41xx module comes pre-loaded with the RTX Platform Firmware which has support for Co-Located Applications. The application developer can then build his/her own Co-Located Application using the API's defined by the Platform Firmware and the CoLA framework. The Application can be loaded into the module for execution without having to modify the rest of the Platform.

3.2.1 Co-Located Application SW Blocks

The different parts of the Co-Located Application are briefly described below.

- **User Application:** This is the component implementing the application functionality of the AR4100P Wi-Fi device. It is normally written by the application programmer / developer using the API described in ([IS1]).

- **Application Framework:** This is a component implementing common application functionality reused by all reference applications except for Blinky.

- **Hardware Drivers:** This is a component implementing common hardware drivers used by all reference applications except for Blinky.

- **Application Protocols:** These are optional product specific functional layers implementing protocols for a specific functionality. The application protocol offloads the application developer by implementing a number of translation protocols like XML coding / decoding for message payloads, parsing of incoming messages and construction of outgoing messages. Application protocol implementations may optionally offer a mail-based API.

- **Networking Applications:** These are optional product-specific functional components, implementing a variety of networking applications such as FTP, TFTP, HTTP, Web server, etc. Networking Applications may be part of the RTX SW delivery as either source code or a binary library. Networking Application implementations may optionally offer a mail-based API.

Application protocols and Networking Applications may be part of the RTX SW delivery as either source code or binary library. Check the RTX41xx download area for SDK updates.

3.2.2 Module Firmware SW Blocks

The different parts of the Module Firmware are briefly described below.

- **Co-Located Application (CoLA) Framework:** This component implements a programming model where the application is dynamically linked with the services provided by the lower layers. The application is compiled and linked as a separate program that at runtime is loaded and run as a task under the Operating System.

- **API**: This is the interface exposed by the Platform Firmware. It exposes all the functionalities needed by the application to implement a Wi-Fi device such as a sensor or an actuator. A detailed description of all the APIs available can be found in ([IS1]). All the APIs are mail-based.

- **Operating System:** RTX low power operating system implementing the necessary functionality to host internal tasks as well as the Co-Located Application.

- **Networking Stack:** This is a functional layer implementing the IP networking stack for IPv4 and IPv6. The networking stack is executed on the AR4100P Wi-Fi chip on RTX41xx.

- **DNS Client:** The DNS client is used to translate domain names to IP addresses by querying a DNS server.

- **HTTP: Common HTTP server and client implementation.** The HTTP server implementation includes TCP connection handling, parsing of HTTP request messages, generation/sending of HTTP response messages, and storing of HTTP resources (WEB pages) represented by path string and a call back function used to generate the content. The HTTP client implementation includes TCP connection handling, generation/sending of HTTP request messages, and parsing of HTTP response messages

- **WiFi Management:** This component handles all aspects of Wi-Fi connection to a Wi-Fi access point, including security and key handling to secure the wireless connection, Wi-Fi power management, etc.

- **Power Management:** This component handles the MCU internal clock trees as well as the power management module. It ensures that any MCU internal part or external peripheral is only running for the appropriate amount of time to preserve power.

- **Firmware Management:** This component implements a functionality to perform a remote firmware update of the Co-Located Application.

- **NVS Management:** This component implements a Non-Volatile Storage (NVS) in a part of the internal FLASH in the MCU.

- **Drivers:** This is a functional layer implementing a number of hardware drivers for MCU peripherals as well as the physical interface to the Wi-Fi sub-component.

3.3 The Application Framework

The application framework, shared by all sample applications except for Blinky, is based on protothreads, facilitating the implementation of anything from simple applications to high complexity state machines. Protothreads are event driven, lightweight and stackless threads, optimized for memory constrained systems such as the RTX41xx.

A protothread remains active until it yields focus to other threads. This happens either when the active thread decides to do so, when it has finished its task, or when it reaches a point where it must wait for some event to occur. Multiple threads can be in a state waiting for system events simultaneously. Events will be distributed to all waiting threads, one at the time, so that each of the threads can react to the event or ignore it as they receive focus. Events are often received in the form of mails. See section 3.4.1 for a description of the mail system.

It is not the purpose of this user guide to describe the inner workings of protothreads. It is recommended to look at some of the sample applications provided with the SDK. Think of protothreads as tasks that execute as linear code until they reach a point where they wait for an event, at which point they yield focus to other threads.

One important thing to keep in mind when it comes to protothreads is that local variables are NOT preserved when focus is yielded to other threads. Variables that must be preserved across such context switches must be declared as global variables.

The nature of protothreads allows the underlying operating system and platform to keep track of the activity level in the application framework. This is an extremely valuable feature because it enables the platform to automatically force the microcontroller into the lowest power mode possible. The application programmer will not have to make special provisions in the code for this to happen. Note that the application must still control the state of the Wi-Fi radio by switching it on and off as required in order to optimize power consumption.

3.4 The RTX OS

The RTX proprietary operating system, ROS, is designed for embedded implementations across a wide variety of applications. The RTX OS is a cooperative OS based on a simple mail or messaging system, allowing individual tasks to communicate with each other across the system. The CoLA framework is running as just one such task. The CoLA task runs in its own thread and will only receive focus when there is a mail for it to process. CoLA applications must therefore be implemented as fully event driven applications.

3.4.1 The ROS Mail system

Each mail sent within the OS is tagged with a descriptive identifier and often a set of parameters. The identifier, referred to as a ROS primitive, is often uniquely representative of some system event or aggregation of events, and the identifier name is normally descriptive of the event or events in question. All tasks have access to the mailing system. This includes the CoLA task.

The CoLA task is able to send a mail to itself. This is useful when one protothread needs to communicate with other waiting protothreads. Sending the simplest type mail, a mail type without parameters, can be achieved as shown below:

```
RosMailSendP0(COLA_TASK, COLA_TASK, <ROS Primitive>);
```

New ROS primitives must be defined in the file RosPrimitiv.h located in the \Projects\Amelie\COLApps\Config\ROS folder of the SDK.

Hardware drivers typically communicate with the application through ROS mails. This happens, for instance, when a buffer is empty or when a transmission has succeeded.

3.4.2 Receiving and acting on ROS Mails

An application running under the CoLA framework will require a function named ColaTask to which system mails are sent. Usually all mails will be distributed to protothreads from here. Blinky, being a minimum implementation, does not rely on protothreads and is the only sample application that does not re-distribute mail. All other sample applications redistribute all mails sent to the ColaTask.

The ColaTask function will typically be pre-screening mails in order to handle at least the initialization and termination of the task. Initialization of the CoLA task usually entails initialization of system hardware and the protothread framework. Redistribution of mail to the pending protothreads is achieved, after the pre-screening, by a call to the function PtDispatchMail():

```
PtDispatchMail(&PtList, Mail);
```

3.4.3 The ROS Timer system

The operating system provides access to a simple timer system from the CoLA framework. The application can basically subscribe to a mail from the OS after a specified time has expired. The application is able to unsubscribe from the timeout mail at any time before it expires. Timer mails share the same TIMEOUT primitive, but can have individual sub-primitives defined in the file RosTimerDef.h located in the \Projects\Amelie\COLApps\Config\ROS folder of the SDK. These sub-primitives are stored in the Timeout.Parameter property of the TIMEOUT mail.

A control structure must be declared in order to set up and use a timer:

```
static const RosTimerConfigType ExampleTimer =
                               ROSTIMER(COLA_TASK, TIMEOUT, EXAMPLE_TIMER);
```

The timer can then be started:

```
RosTimerStart(APP_LED_TIMER, 10 * RS_T1SEC, &ExampleTimer);
```

The CoLA task will receive a ROS mail after the timer expires – in the above example, 10 seconds after the timer was started.

3.4.4 Non-volatile Storage (NVS)

The operating systm provides access to the NVS of the entire system, but a specific area is reserved for CoLA usage. The offset and the size of this are of the NVS is available in the two variables `ColaNvsOffset` and `ColaNvsSize`. The NVS can be accessed through the `NvsRead()` and the `NvsWrite()` functions.

An example of how to read byte 7 within the CoLA area:

```
rsuint8 b;
NvsRead(ColaNvsOffset+7, sizeof(b), &b);
```

3.5 Application Reserve

The flash / RAM space left for the application depends on the configuration of the protocol and stack. Please refer to the RTX4100 and RTX4140 datasheet for more information ([DS1]) and ([DS2]).

3.6 Hardware resources

The hardware resources/peripherals of the Energy Micro MCU used by the module firmware are outlined in this section. The application should not use any of these hardware resources unless the module FW offers support to disable the usage of the hardware resource/peripheral.

The CoLA can access the internal HW block not used by the platform through the "emlib" provided by Energy Micro. Source code for the emlib can be found in the 3Party folder where the SDK is installed. The emlib is included as source in the build of the CoLA.

The following HW resources are used by the module FW:

- **LEUART1** – Used for RTX EAP UART interface running 9600 BAUD. The module pins PC7 and PC6 are occupied by the LEUART1. Can be disabled in the module FW and used from the CoLA.

- **USART1** – Used for RTX EAP UART interface running 115200 BAUD. The module pins PA4/PC0 and PA3/PC1 are occupied by the USART1. Can be disabled in the module FW and used from the CoLA. This USART is used by default for CoLA download with the Cola Controller.

- **USART0** – Used for SPI communication between the EnergyMicro chip and the AR4100 WiFi chip. The EnergyMicro pins PE9, PE10, PE11, PE12 and PE13 are routed internally in the RTX4100 module and are not exposed on the module pin out.

- **GPIO ISR** – The GPIO interrupt handling is done by the module FW and an API mail is sent to the CoLA task when a GPIO interrupt is detected.

- **GPIO's** – The EnergyMicro pins PF5, PA5 and PA6 are used internally in the module and should not be changed by the application. PB13 can be used by the module FW to enable an external DC/DC voltage step-up on 2 cell designs. This feature must be enabled from the CoLA by an API mail if required. The GPIO's available to the Application can be seen from the RTX4100 module data sheet taking the above restrictions in mind.

- **Clock management** – The module FW controls both the HF and LF clock. The HF clock is generated by the internal HF oscillator set to 28 MHz, and the LF clock is generated by the internal LF oscillator running 32.768 kHz. The module FW makes sure that the HF clock is disabled when it is not needed. It is not possible to control the clock configuration from the CoLA with the current module FW.

- **RTC** – The Real Time Counter is used to drive the timer system in the RTX OS in the module FW. The RTC must not be reconfigured from the CoLA, but it can be used as clock reference if needed. Compare channel 0 is used to trigger timer tick interrupt, but compare channel 1 is not used by the module FW.

- **Cortex-M3 SysTick** – The SysTick inside the Cortex-M3 CPU is used by the module FW to implement the "performance counter" (RsReadPerformanceCounter()) that can be used to do performance timing with 28 MHz resolution.

- **Debug Interface** – The pins PF0 and PF1 can be used to connect a debugger like e.g. the RTX2040 debugger to the EnergyMicro chip. PF0 and PF1 can be used as GPIO if the debug interface is disabled.

3.7 SDK Folder Structure

This section describes some of the most important directories of the SDK.
It is assumed that the SDK has been installed in c:\AmelieSdk\vX.XX, and the following description of the folder structure is relative to this location.

The following folders are at top level:

- **3Party** – includes source files from 3rd parties. The most important file found here is the "efm32lib" from Energy Micro which is used to access the peripherals of the MCU.

- **ColaController** – The PC tool used to download Co-Located Applications to the module via the UART.

- **Components** – Source code to some common components used when the CoLA is built.

- **Documents** – Documentation folder.

- **Include** – Common header files.

- **Projects** – This is where you find the reference CoLA applications

- **Tools** – A set of tools used when the CoLA is built. This also includes the RtxBuild system used for the build process.

In the Project folder (Projects\Amelie\COLApps) you find the following folders:

- **Apps** – A set of reference applications
  - Blinky – The source code for the Blinky application
  - Blinky/Build/RTX4100_WSAB – Build folder for RTX4100 module on WSAB.
  - Blinky/Build/RTX4140_WSAB – Build folder for RTX4140 module on WSAB.
  - …

- **Config** – This folder holds various configuration header files required to build the reference applications.

- **Rsx** – PC tool for API mail trace via UART.

- **Scripts** – build/linker scripts.

In the Components folder (Projects\Amelie\Componets) you find the following folders:

- **Api** – Some API header files

- **ApiMps** – API mail packing and sending helper functions

- **Drivers** – Implementation of various hardware drivers

- **PtApps** – Protothread based application framework used by all reference applications except Blinky.

4 Application development

Application development can be roughly separated into three major tasks:

- Editing and/or modifying source code
- Building or Compiling and linking the application
- Loading the application onto the module

The first step, editing source code, can be managed through any text editor, but a c-editor makes the task easier. There are free editors available such as Notepad++ (http://notepad-plus-plus.org)

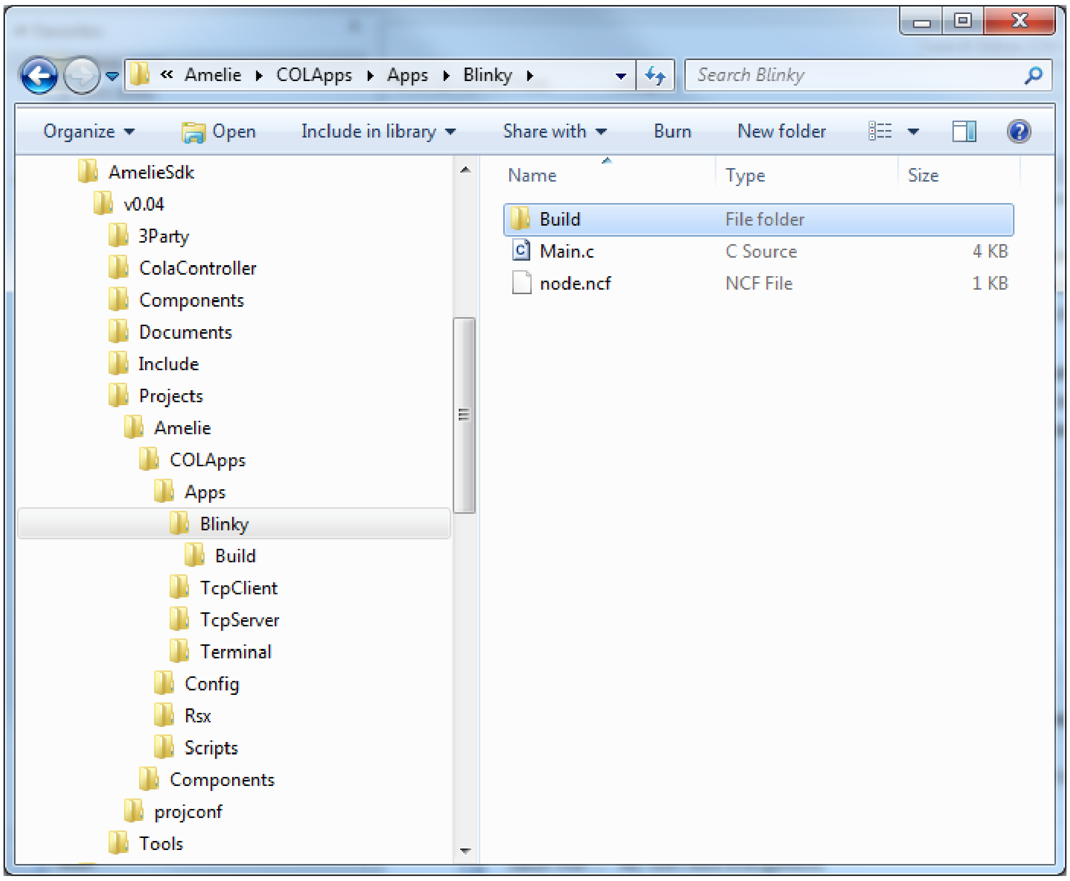
Building the code requires a compiler and linker. The Amelie SDK relies on ARM's GCC Embedded tool chain from Launchpad (see the Tools Installation Guide [UG3]

Loading the compiled application onto the platform is achieved through the use of the Windows application CoLA Controller that came with the SDK.

In this section, we will address the details of these 3 steps, using the minimum application Blinky as an example. It is assumed that the above mentioned tools all have been successfully installed at this time and that the RTX4100 DVK is used during the process.

4.1 Editing Source Code

The SDK sample applications are all organized as shown below.



```
AmelieSdk
  v0.04
    3Party
    ColaController
    Components
    Documents
    Include
    Projects
      Amelie
        COLApps
          Apps
            Blinky
              Build
            TcpClient
            TcpServer
            Terminal
          Config
          Rsx
          Scripts
        Components
      projconf
    Tools
```

Name	Type	Size
Build	File folder	
Main.c	C Source	4 KB
node.ncf	NCF File	1 KB

The screen shot shows the Blinky project folder and its location. The source code is found in Main.c. The other file node.ncf contains a list of all the source files that must be compiled when Blinky is built. In the existing file, there is only one such source file listed: Main.c.

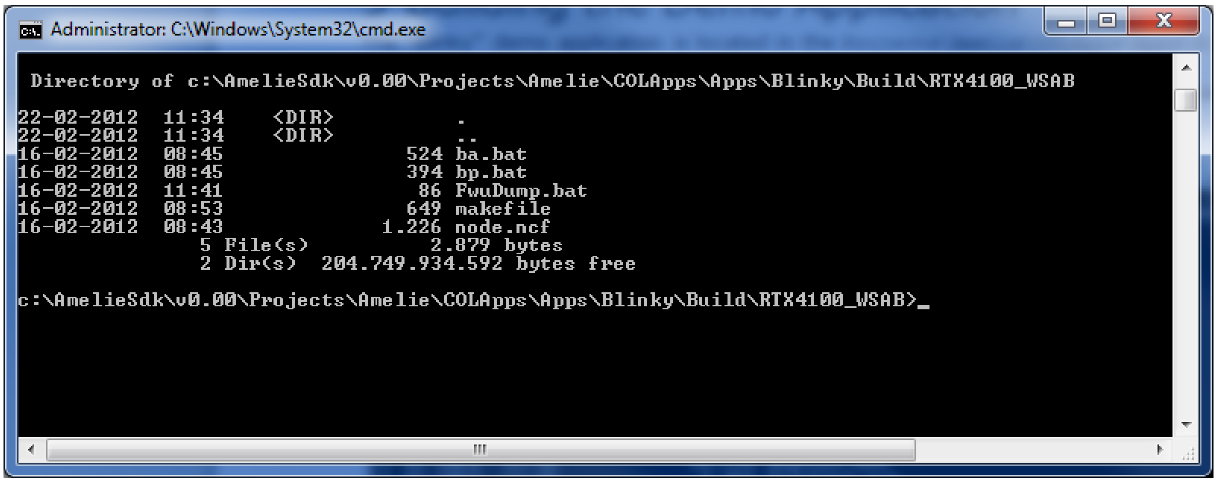
Adding one or more files to the project requires editing of the node.ncf file. The source files must be stored in the same folder as Main.c. The listing below shows an example of node.ncf after a file Foo.c has been added to the project:

```
###############################################################################
#                          NODE CONFIGURATION FILE
# Please refer to the PerlBuild documentation for information regarding the
# meaning of the fields in this file.
###############################################################################
NODE_NAME=Blinky
NODE=Projects\Amelie\COLApps\Apps\Blinky
C-FILES=
  Main.c
  Foo.c
```

After source files have been edited, they must be saved before the build process is started.

4.2 Building the application

To build a project, open a Windows command prompt, and navigate to the projects Build\RTX4100_WSAB folder (use the Build\RTX4140_WSAB folder for RTX4140). Run the batch command ba.bat (BA: Build All) by typing 'ba' into the command prompt.



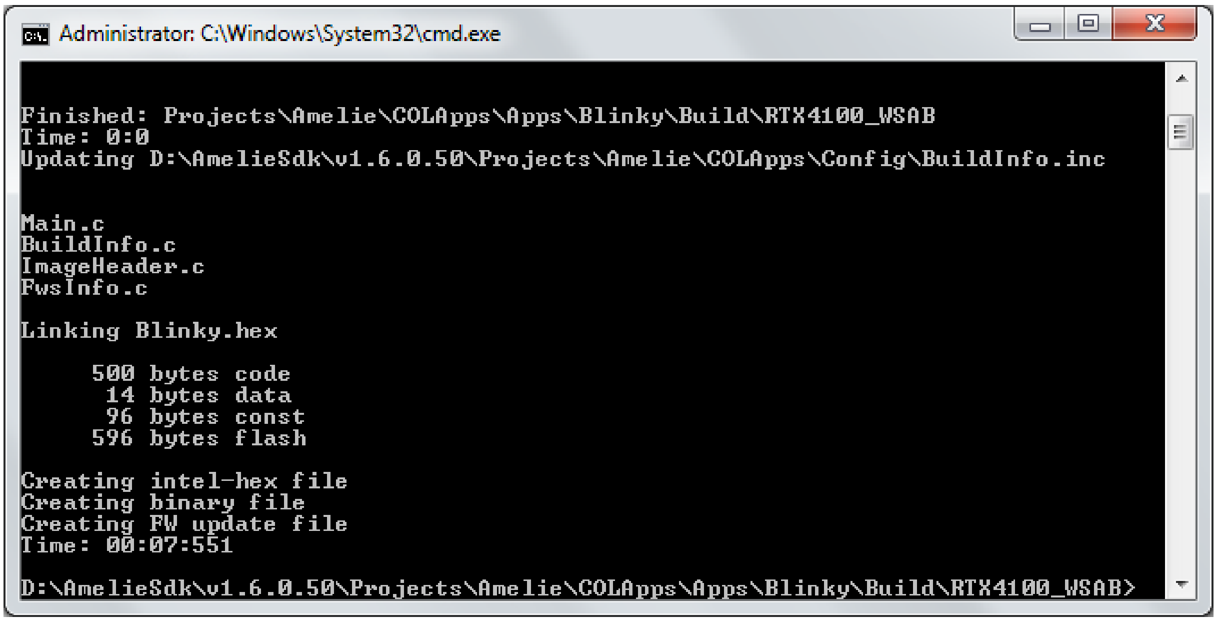
```
 Directory of c:\AmelieSdk\v0.00\Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4100_WSAB

22-02-2012  11:34    <DIR>          .
22-02-2012  11:34    <DIR>          ..
16-02-2012  08:45               524 ba.bat
16-02-2012  08:45               394 bp.bat
16-02-2012  11:41                86 FwuDump.bat
16-02-2012  08:53               649 makefile
16-02-2012  08:43             1.226 node.ncf
               5 File(s)          2.879 bytes
               2 Dir(s)  204.749.934.592 bytes free

c:\AmelieSdk\v0.00\Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4100_WSAB>_
```

The build should start if the tool chain has been installed properly. If this step fails, repeat the procedures in ([UG3]), Tools Installation.

After Blinky has compiled and linked correctly, the result should look like this:



```
Finished: Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4100_WSAB
Time: 0:0
Updating D:\AmelieSdk\v1.6.0.50\Projects\Amelie\COLApps\Config\BuildInfo.inc


Main.c
BuildInfo.c
ImageHeader.c
FwsInfo.c

Linking Blinky.hex

     500 bytes code
      14 bytes data
      96 bytes const
     596 bytes flash

Creating intel-hex file
Creating binary file
Creating FW update file
Time: 00:07:551

D:\AmelieSdk\v1.6.0.50\Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4100_WSAB>
```

The output from the entire build process can be found in the text file BuildLog.txt that can be accessed from any text editor.

4.2.1 Loading

IMPORTANT: From RTX41xx SDK 1.6.x.x the CoLA controller tools have changed to support a new firmware file content with extension .fws instead of .fwu.
It is very important that you use the CoLA controller installed with SDK version 1.6.x.x as the previous version of the CoLA controller will not support .fws files.

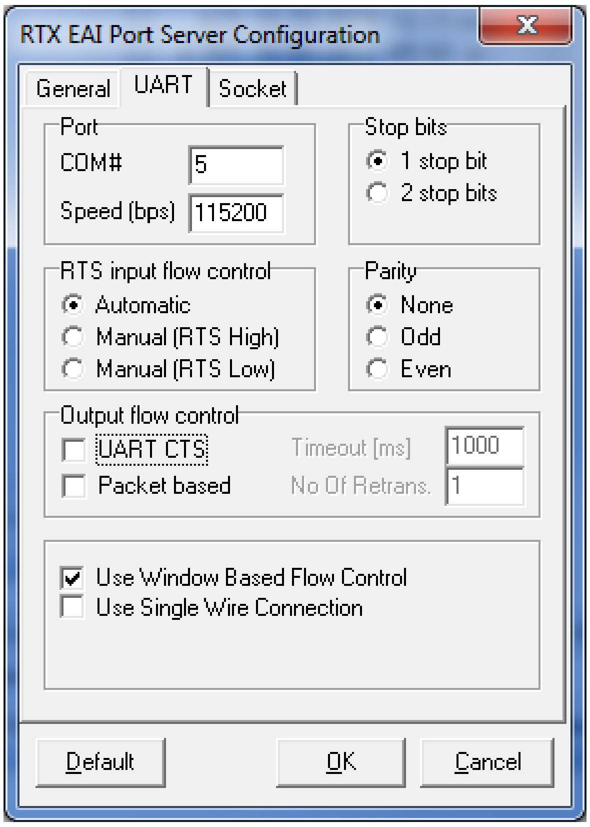
This section lists the steps needed to load the pre-built *Blinky* example into the hardware and run it. You can do it either using the CoLA Controller application, or the Nexus 4 application if you have a RTX2040 Unity-II box.

Mount the WSAB in the Docking Station and connect it to a USB port on your PC.
Using COLA Controller

1) From the start menu, locate the *Amelie Sdk* group and run the *COLA Controller* application.

2) (First time only): Right-click the *RTX EAI Port Server* icon that is shown in the task bar. Select *Setup*. Select *UART Transport Layer*.

3) IMPORTANT - UART configuration for platform version **1.2.0.x (and newer):**
   a. Enter the COM port number assigned to the Lower COM # (US1 for USART) of the Docking Station. It should be identical to the Virtual COM port identified during installation [2].
   b. Select *115200* bps *Speed*
   c. Check *Use Window Based Flow Control*
   d. Uncheck *Packet Based* output flow control and *UART CTS.*


RTX EAI Port Server Configuration

General | UART | Socket

Port
COM# 5
Speed (bps) 115200

Stop bits
● 1 stop bit
○ 2 stop bits

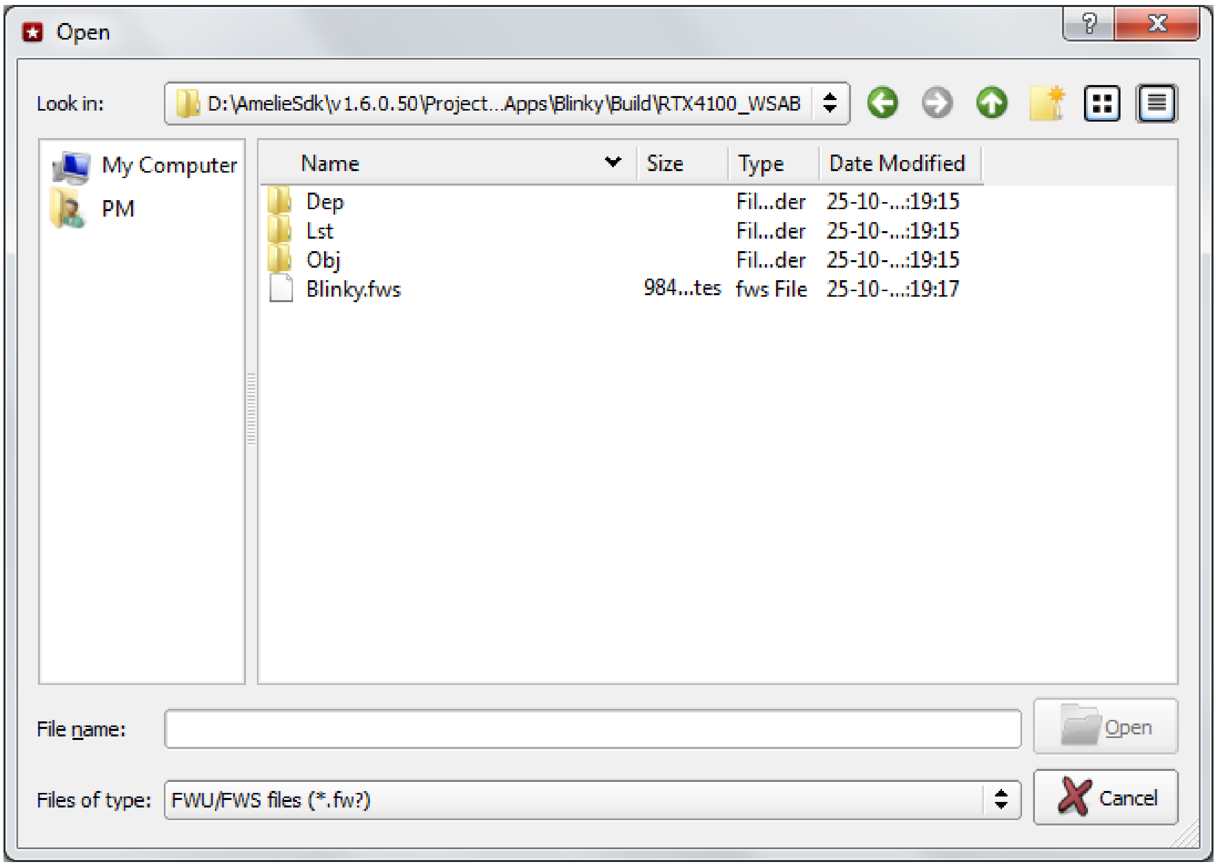
RTS input flow control
● Automatic
○ Manual (RTS High)
○ Manual (RTS Low)

Parity
● None
○ Odd
○ Even

Output flow control
☐ UART CTS — Timeout [ms] 1000
☐ Packet based — No Of Retrans. 1

☑ Use Window Based Flow Control
☐ Use Single Wire Connection

4) Click the *Refresh* button in the *COLA Controller* application. The image list is re-freshed
5) Click the browse button, and select the file
RTX4100:
Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4100_WSAB\Blinky.fws.
RTX4140:
Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4140_WSAB\Blinky.fws.


Open

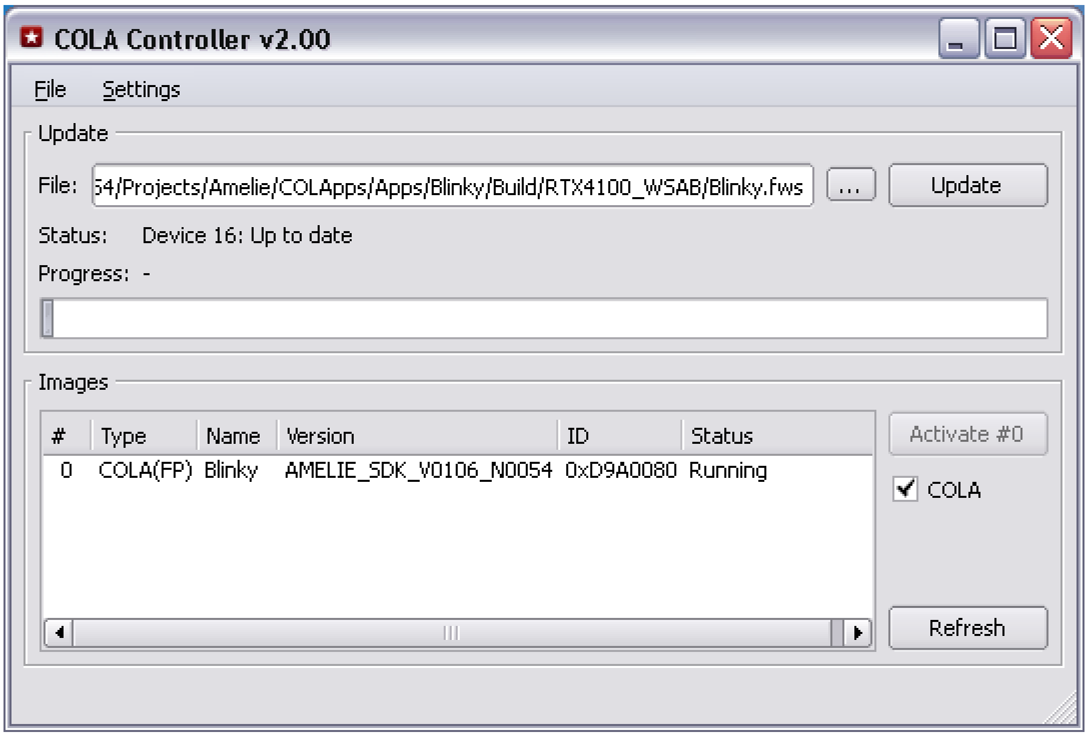
Look in: D:\AmelieSdk\v1.6.0.50\Project...Apps\Blinky\Build\RTX4100_WSAB

My Computer
PM

Name	Size	Type	Date Modified
Dep		Fil...der	25-10-...:19:15
Lst		Fil...der	25-10-...:19:15
Obj		Fil...der	25-10-...:19:15
Blinky.fws	984...tes	fws File	25-10-...:19:17

File name:
Files of type: FWU/FWS files (*.fw?)

6) Click the *Update* button.



File: 54/Projects/Amelie/COLApps/Apps/Blinky/Build/RTX4100_WSAB/Blinky.fws

Status: Device 16: Up to date

Progress: -

#	Type	Name	Version	ID	Status
0	COLA(FP)	Blinky	AMELIE_SDK_V0106_N0054	0xD9A0080	Running

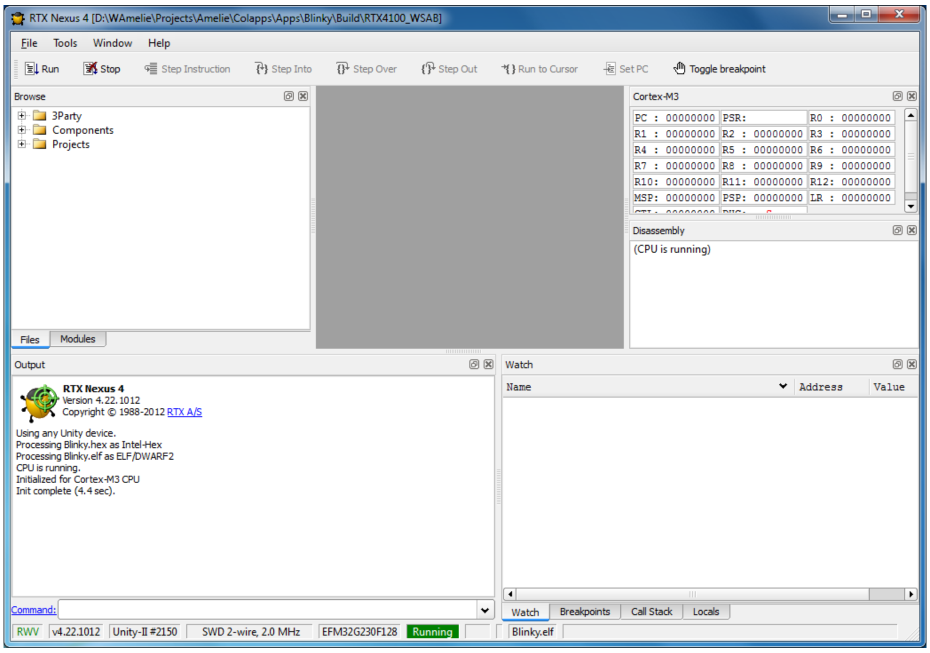
☑ COLA

7) When the update is completed, press *Refresh* to update the image list. If the status for Blinky shows disabled, then click image 0 (button shows *Activate #0)*, make sure *COLA* is checked, and click the *Activate* button.

Observe the Version number, which should reflect the link date.
The Blinky application is now executing in the target, the LED should be blinking.
Using the RTX2040 Unity-II box
If you have a RTX2040 Unity-II Box, you can use the application called Nexus 4 provided with it.

NOTE: The RTX2040 Unity II is an additional product that is not included in the DVK, you need to purchase it separately. Please contact RTX sales department (sales@rtx.dk) or your distributor.

1) Open the application Nexus 4 and you will see the following window:



RTX Nexus 4 [D:\WAmelie\Projects\Amelie\Colapps\Apps\Blinky\Build\RTX4100_WSAB]
File Tools Window Help
Run Stop Step Instruction Step Into Step Over Step Out Run to Cursor Set PC Toggle breakpoint
Browse
3Party
Components
Projects
Files Modules
Cortex-M3
Disassembly
Output
Watch
Name Address Value
Command:
Watch Breakpoints Call Stack Locals

```
PC : 00000000 PSR:          R0 : 00000000
R1 : 00000000 R2 : 00000000 R3 : 00000000
R4 : 00000000 R5 : 00000000 R6 : 00000000
R7 : 00000000 R8 : 00000000 R9 : 00000000
R10: 00000000 R11: 00000000 R12: 00000000
MSP: 00000000 PSP: 00000000 LR : 00000000
```

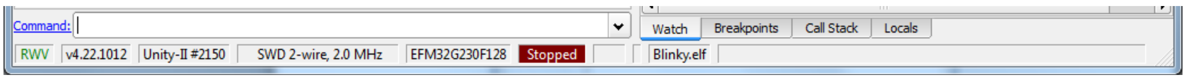
(CPU is running)

```
RTX Nexus 4
Version 4.22.1012
Copyright © 1988-2012 RTX A/S

Using any Unity device.
Processing Blinky.hex as Intel-Hex
Processing Blinky.elf as ELF/DWARF2
CPU is running.
Initialized for Cortex-M3 CPU
Init complete (4.4 sec).
```

RWV | v4.22.1012 | Unity-II #2150 | SWD 2-wire, 2.0 MHz | EFM32G230F128 | Running | Blinky.elf

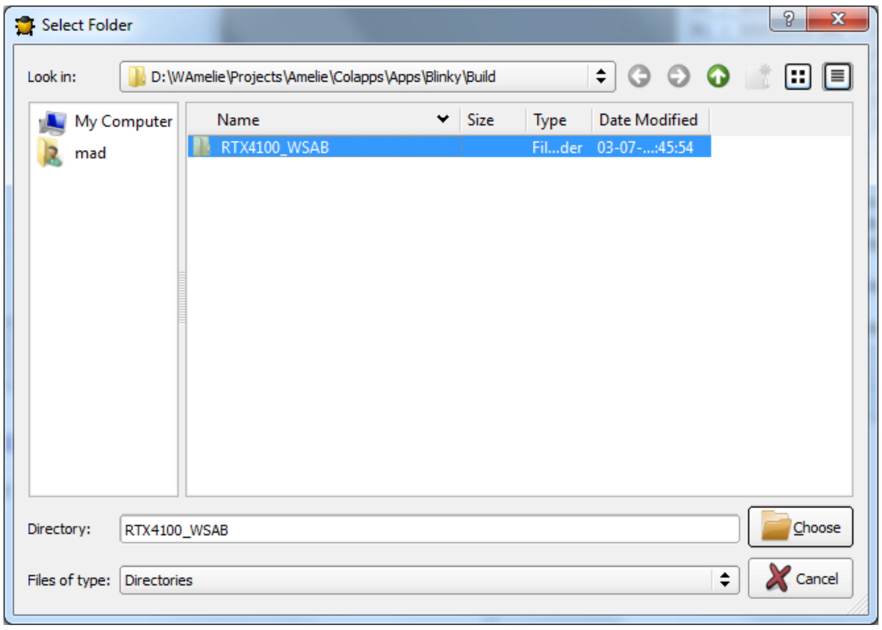
2) In the "Command" line (red arrow), type 'rd' to *reset* and enter the *debug* mode. The status bar will show a red label "*Stopped*".
<u>Note:</u> you cannot load an application if the module is Running.



RWV | v4.22.1012 | Unity-II #2150 | SWD 2-wire, 2.0 MHz | EFM32G230F128 | Stopped | Blinky.elf

3) Go to *File* > *Change Folder*, select the folder
RTX4100: *Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4100_WSAB*
RTX4140: *Projects\Amelie\COLApps\Apps\Blinky\Build\RTX4140_WSAB*

where you recently built your application (see Section 5), then click on *Choose*.



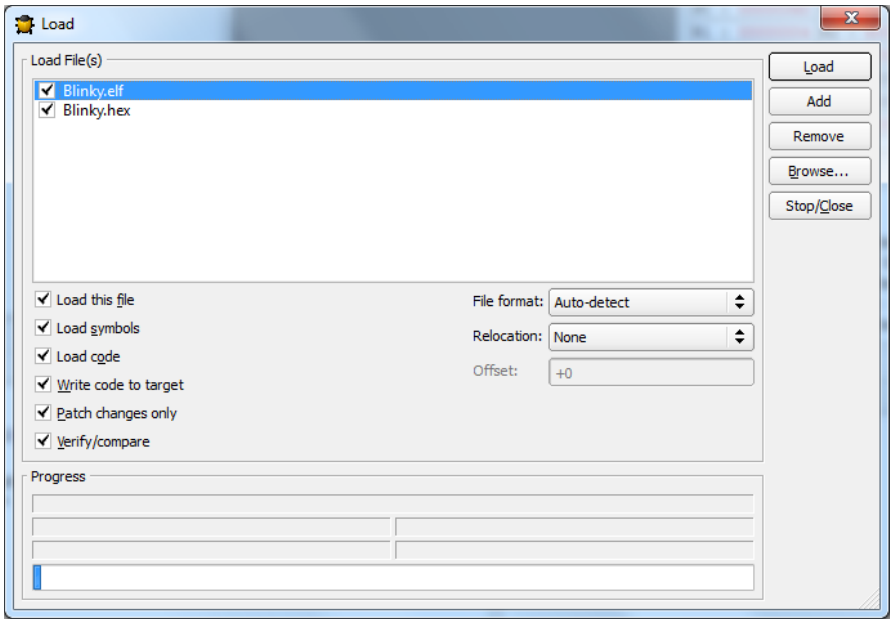
Look in: D:\WAmelie\Projects\Amelie\Colapps\Apps\Blinky\Build

Name	Size	Type	Date Modified
RTX4100_WSAB		Fil...der	03-07-...:45:54

Directory: RTX4100_WSAB
Files of type: Directories

4) Go to *File* > *Load* (or Enter Ctrl+D). In the new window, select *Browse...*

Select the files with the extension *.elf* and *.hex*, then click on *Load*.



☑ Blinky.elf
☑ Blinky.hex

☑ Load this file
☑ Load symbols
☑ Load code
☑ Write code to target
☑ Patch changes only
☑ Verify/compare

File format: Auto-detect
Relocation: None
Offset: +0

5) Select the command line again, and type 'r' to reset your module.

Now, the led should be blinking on your RTX4100 module.

At this stage you are ready to develop your own applications taking advantage of the Wi-Fi capabilities of the RTX4100 module.

5 Reference applications

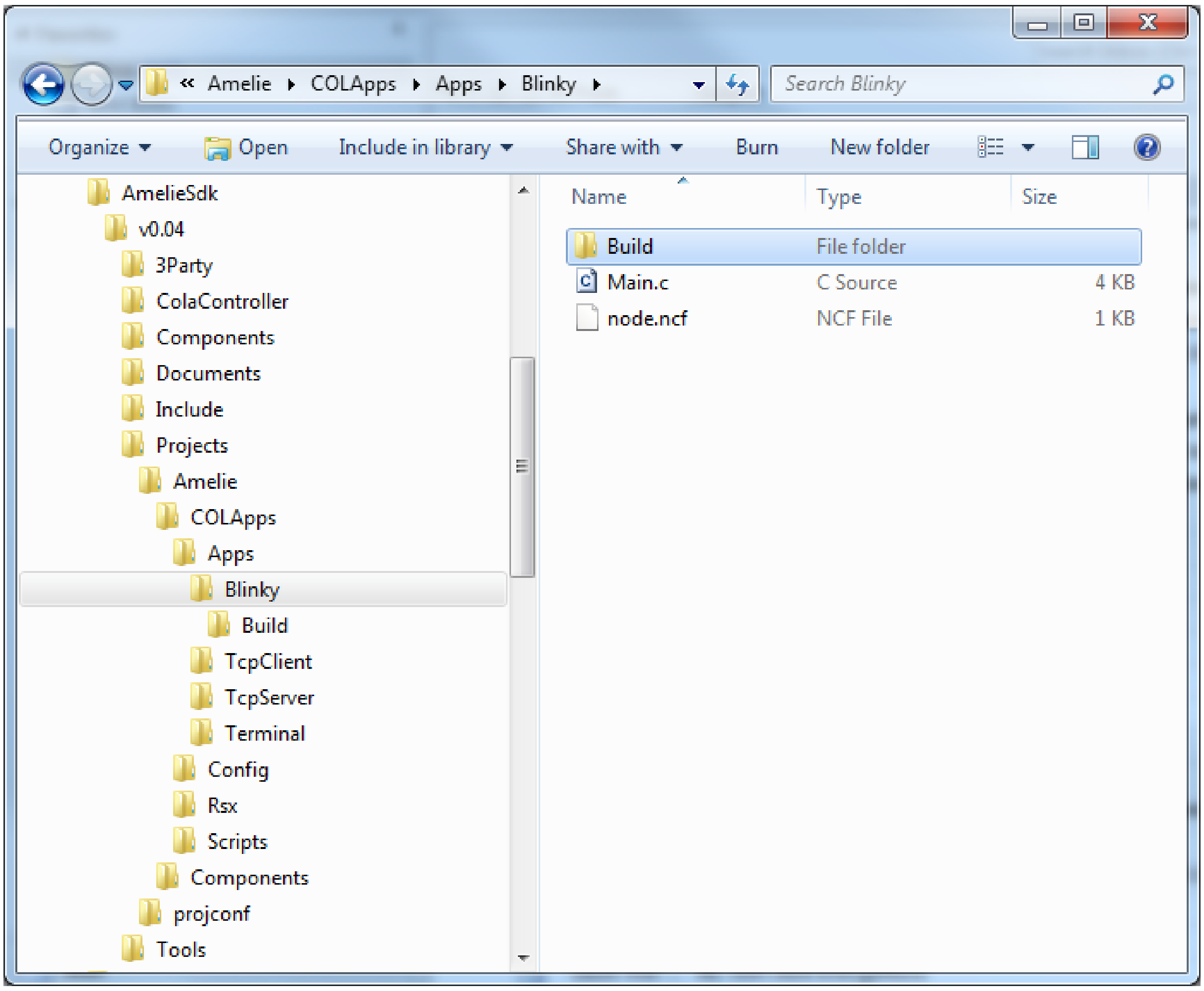
Some reference applications are included in the SDK to show how different applications can be implemented using the CoLA interface and the API of the Amelie platform. A short description of all the reference applications is given in this section.

The first reference application described is called Blinky; which is an application that uses a timer to blink with the green led. Blinky is very simple, but serves as a good example to show the concept of the CoLA framework, and how the CoLA task is integrated with the RTX OS in the module FW. We use the Blinky application to illustrate how an application is built, how to add new mail primitives, and how to add timers to the system. It should be straight forward to apply this to the other reference applications described afterwards.

The other reference applications show how to implement more complex applications using the Application Framework (PtApps). These applications are not described in details, but are focused on giving an overview of the application functionality. All the details can be seen in the source code of the applications. All the reference applications are written to demonstrate a few features each and they are written in a way that should make them a good starting point for new application development.

5.1 Blinky

Blinky is a very simple application that does only one thing: blink with the green led, triggered by timeouts. This makes Blinky the ideal application to dissect line for line in order to explain how CoLA code is properly constructed. Assuming the SDK has been installed on the C drive, navigate to the “Blinky” folder below, and open the Main.c file using your favourite C code editor.



```
AmelieSdk
  v0.04
    3Party
    ColaController
    Components
    Documents
    Include
    Projects
      Amelie
        COLApps
          Apps
            Blinky
              Build
            TcpClient
            TcpServer
            Terminal
          Config
          Rsx
          Scripts
        Components
      projconf
    Tools
```

Name	Type	Size
Build	File folder	
Main.c	C Source	4 KB
node.ncf	NCF File	1 KB

5.1.1 Overview of “Blinky”

The Main.c file includes a number of header files for definitions. These are:

```
#include <Core/RtxCore.h>
```
Standard RTX definitions of types etc.

```
#include <Ros/RosCfg.h>
```
Definitions of project specific setup for the ROS (RTX Operating System).

#include <PortDef.h>

```
#include <Cola/Cola.h>
```
Prototypes for function calls to access the API functions through the CoLA handler.

```
#include <efm32_gpio.h>
```

Definitions of GPIO ports on the Energy Micro EFM32 MCU.

The simple program performs the following:

1. Receives INITTASK from the RTX Operating System.
    a. Sets the GPIO driving the LED as output.
    b. Starts a 1 second timer with APP_LED_TIMER as parameter.
2. Receives TIMEOUT from the RTX Operating System (Timer started above)
    a. Toggles GPIO driving LED.
    b. Re-starts the timer with 500 ms timeout
3. Receive TERMINATETASK from the RTX Operating System.

INITTASK: First Mail received after start-up

Set LED GPIO pin
Start 1 second timer with APP_LED_TIMER as parameter. All other sample applications initialize hardware, and protothreads here.

```
void ColaTask(const RosMailType* Mail)
{
  switch (Mail->Primitive)
  {
    case INITTASK:
      GPIO_PinModeSet(LEDPORT, LEDPIN, gpioModePushPull, 1);
      RosTimerStart(APP_LED_TIMER, 1 * RS_T1SEC, &AppLedTimer);
      break;

    case TERMINATETASK:
      RosTaskTerminated(ColaIf->ColaTaskId);
      break;

    case TIMEOUT:
      switch (Mail->Timeout.Parameter)
      {
        case APP_LED_TIMER:
          GPIO_PinOutToggle(LEDPORT, LEDPIN);
          RosTimerStart(APP_LED_TIMER, 5 * RS_T100MS, &AppLedTimer);
          break;
      }
      break;

    default:
      break;
  }

}
```

TERMINATETASK:
Received when System OS shuts down

Not used in Blinky, but just shows proper handling

TIMEOUT:
Received when timer started in INIT-TASK expires
Toggle GPIO pin driving LED
Start 500 ms timer again

Redistribution of ROS mail:
All other sample applications will redistribute the ROS mail to the pending protothreads here by calling the `PtDispatchMail()` function.

5.2 The TCP Client and Server Demo

The TCP Client and Server demo applications are more advanced applications demonstrating the use of Wi-Fi connection.

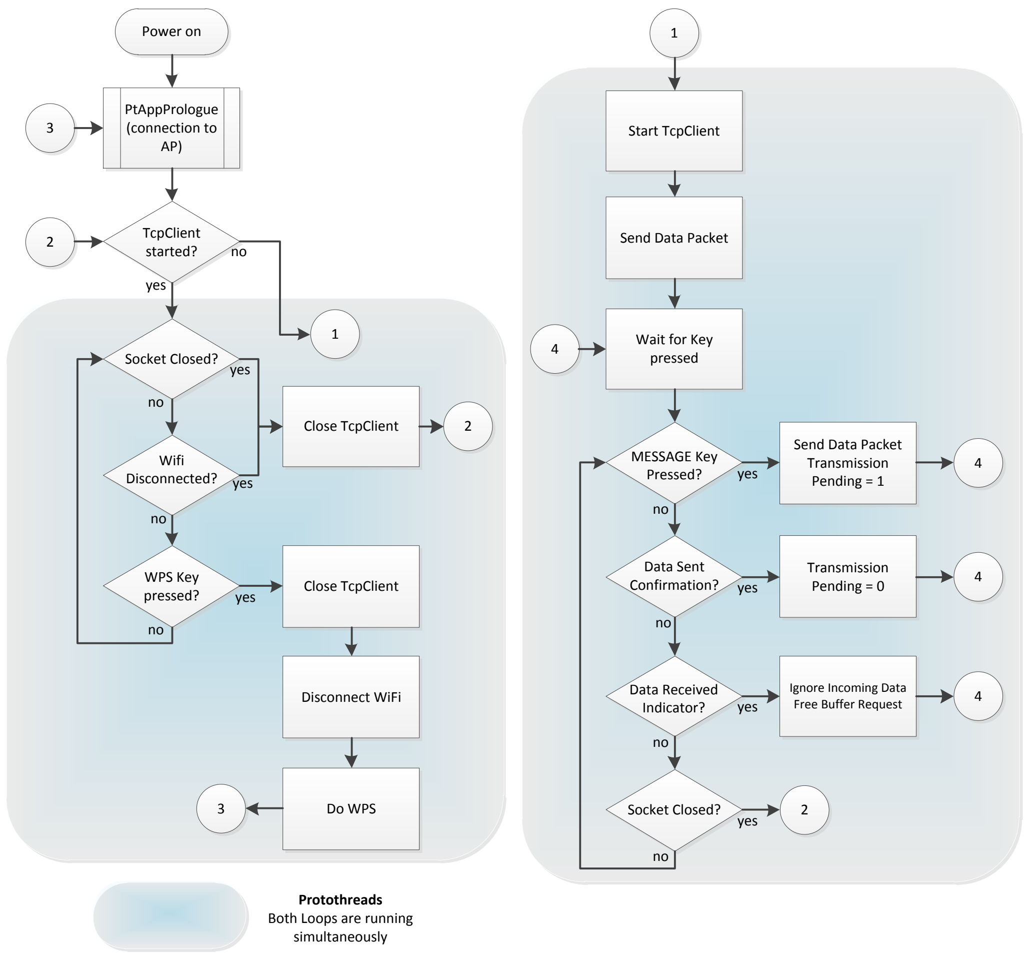
The TCP client and server uses common application code for all CoLA applications, which can be found under: \AmelieSdk\vX.X.X.X\Projects\Amelie\Components\PtApps\

5.2.1 TCP Client

The TCP Client is located under:
\AmelieSdk\vX.X.X.X\Projects\Amelie\COLApps\Apps\TcpClient\

It is a simple TCP client implementation that does the following:

1. Performs a WPS if the 'SW2' button on the WSAB is pressed.
2. Connects to Wi-Fi access point and enables DHCP.
3. Connects to a TCP server (hardcoded address in source code).
4. Sends a dummy data packet once the TCP connection to the server has been established.
5. Sends the dummy data packet again if the 'SW1' button on the WSAB is pressed.

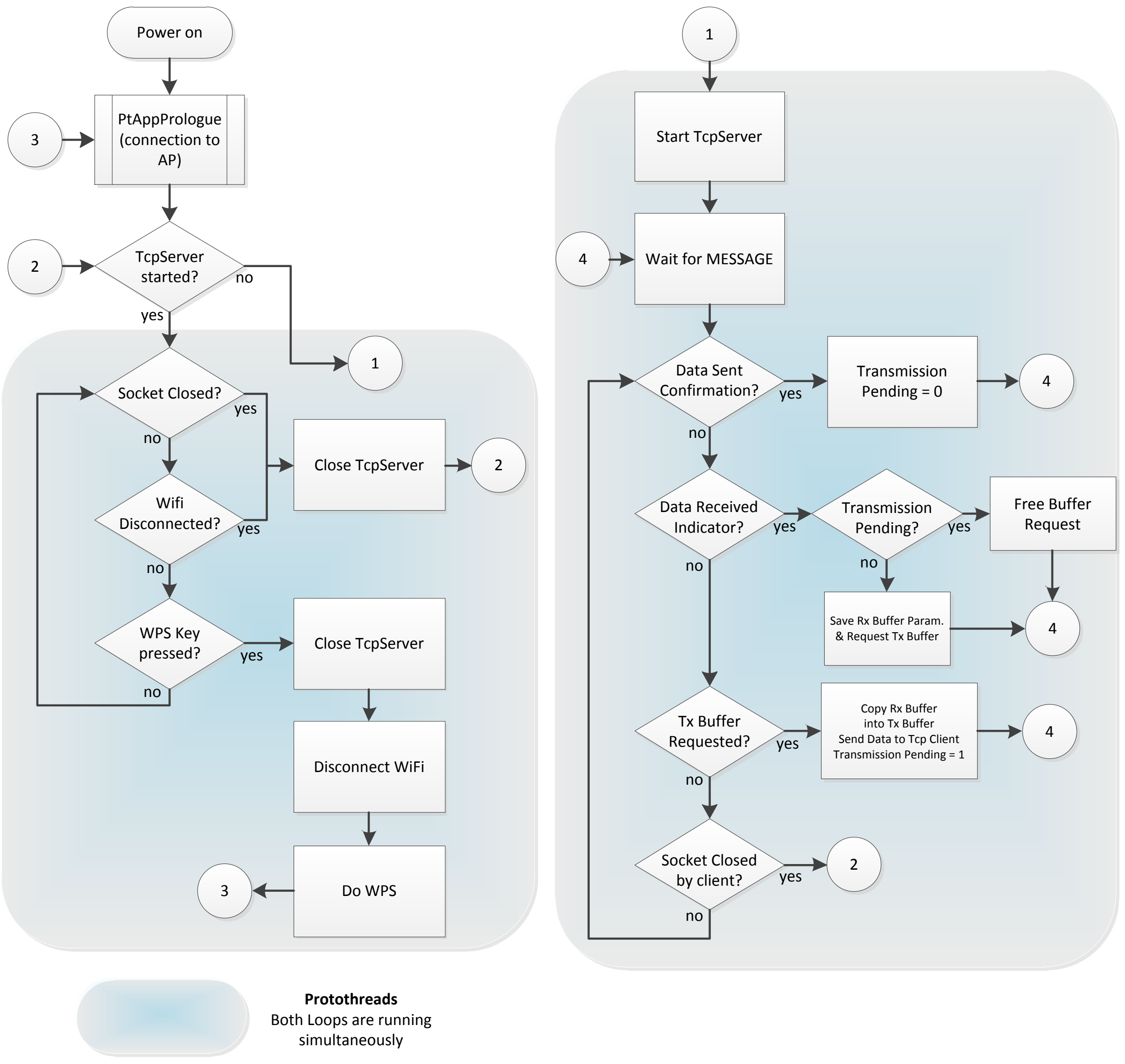


5.2.2 TCP Server

The TCP Client is located under:
\AmelieSdk\vX.X.X.X\Projects\Amelie\COLApps\Apps\TcpServer\

It is a simple TCP server implementation that does the following:

1. Performs a WPS if the 'SW2' button on the WSAB is pressed.
2. Connects to Wi-Fi access point and sets static IP address (defined in source code).
3. Listens for TCP connect (port number defined in source code)
4. Accept connection from TCP client
5. Toggles the green LED on the WSAB each time a data packet is received from the client connected. The data received from the TCP client is looped back to the client.



5.3 The UDP Client and Server Demo

The UDP Client and Server demo applications are more advanced applications demonstrating the use of Wi-Fi connection.

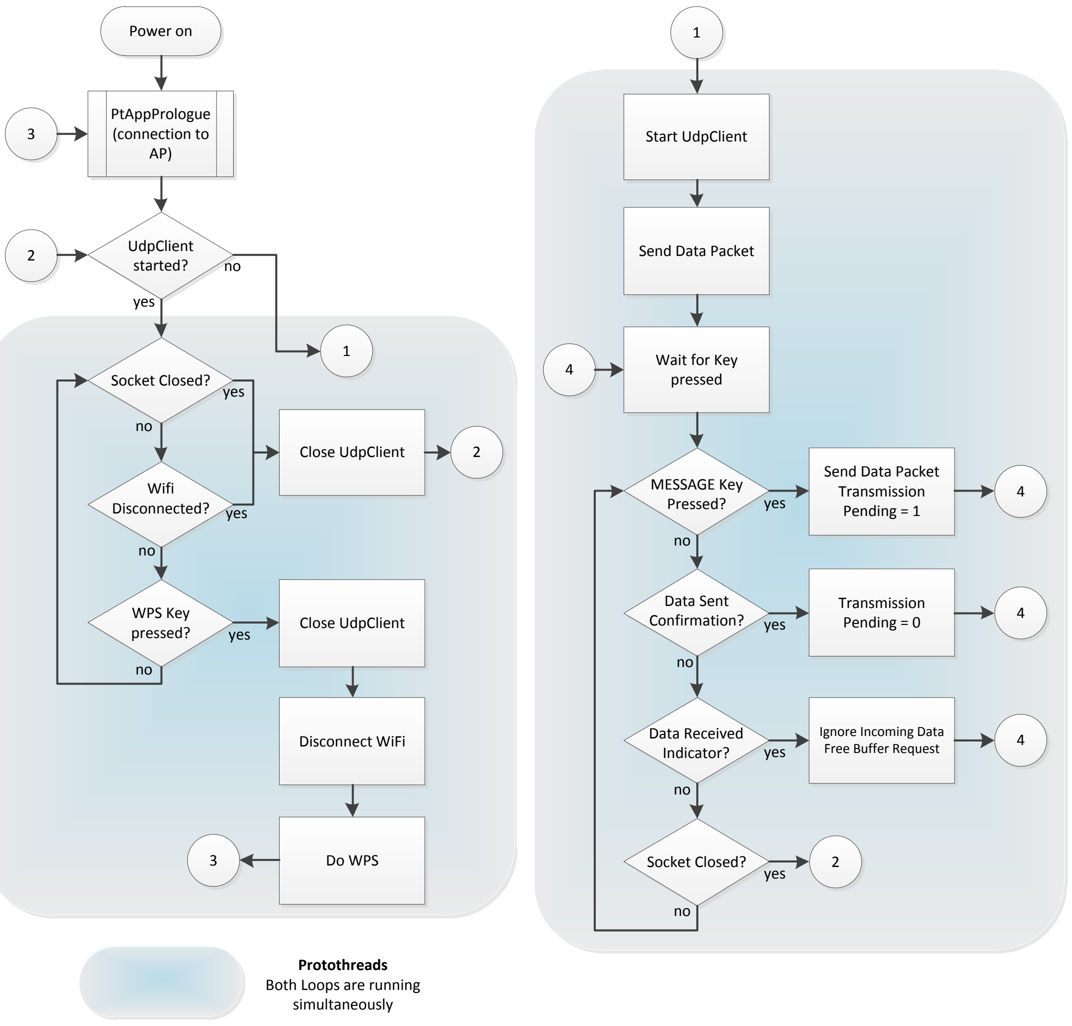
The UDP client and server uses common application code for all CoLA applications, which can be found under:
\AmelieSdk\vX.X.X.X\Projects\Amelie\Components\PtApps\

5.3.1 UDP Client

The UDP Client is located under:
\AmelieSdk\vX.X.X.X\Projects\Amelie\COLApps\Apps\UdpClient\

It is a simple UDP client implementation that does the following:

1. Performs a WPS if the ‘SW2’ button on the WSAB is pressed.
2. Connects to Wi-Fi access point and enables DHCP.
3. Sends a dummy UDP data packet once the connection to the access point has been established.
4. Sends the dummy data packet again if the ‘SW1’ button on the WSAB is pressed.

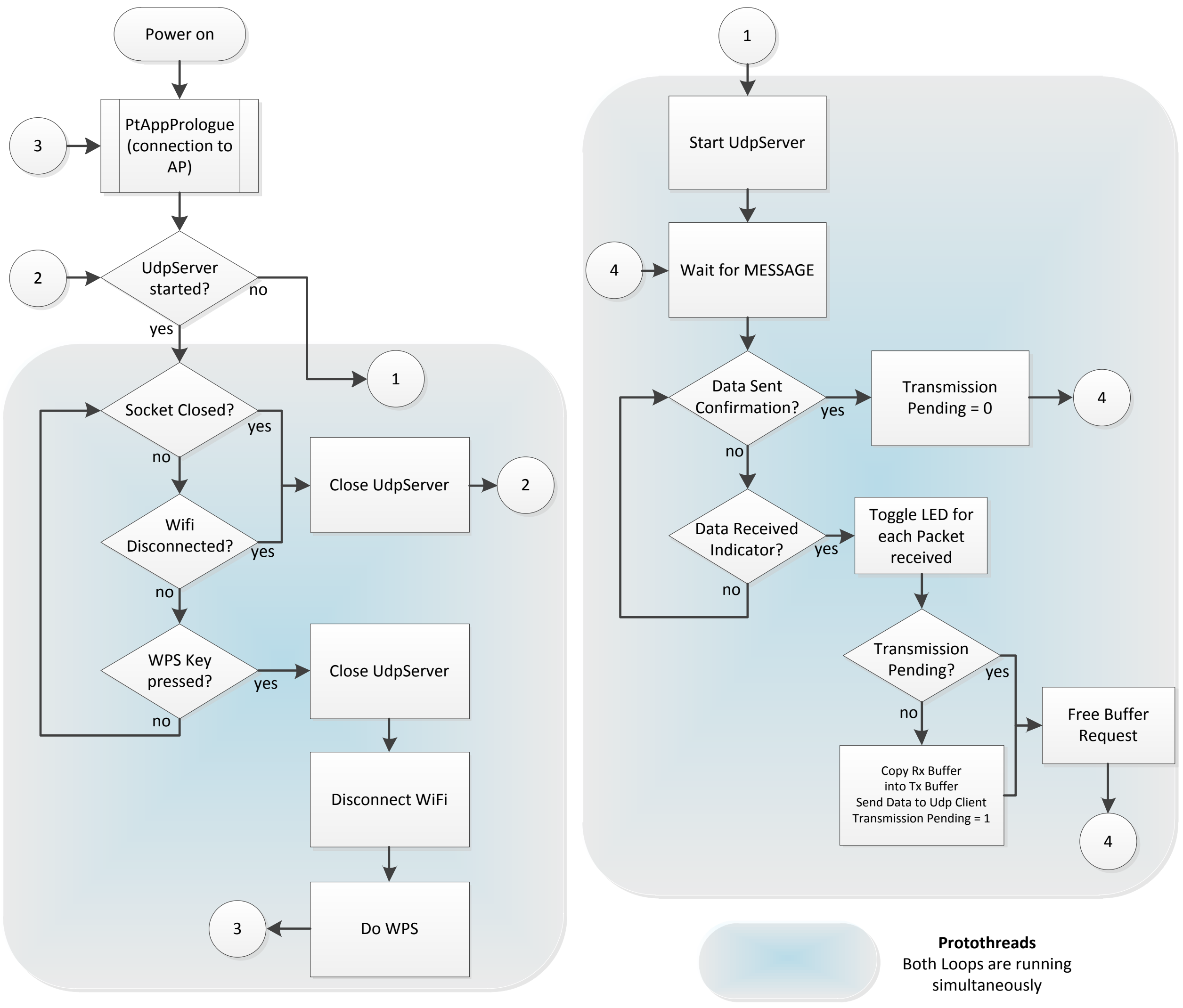


5.3.2 UDP Server

The UDP Client is located under:
\AmelieSdk\vX.X.X.X\Projects\Amelie\COLApps\Apps\UdpServer\

It is a simple UDP server implementation that does the following:

1. Performs a WPS if the 'SW2' button on the WSAB is pressed.
2. Connects to Wi-Fi access point and sets static IP address (defined in source code).
3. Listens for UDP data
4. Toggles the green LED on the WSAB each time a UDP data packet is received. The data received is looped back to the originating address.



5.4 The Terminal Demo Application

The Terminal Demo Application implements a simple shell, handling various commands for Wi-Fi configuration, TCP client/server test, UDP client/server test, etc.

It is identical to the application pre-loaded in the WSAB in the DVK. A more detailed description of the commands supported can be found in ([UG1]).

5.5 The TempSensor Demo Applications

The TempSensor Demo Applications are samples of an advanced application to demonstrate a real case scenario of the RTX4100 module.

The purpose of these applications is to make use of the low power configuration of the module while monitoring the Temperature in a given environment, and report it using Wi-Fi.
There are three different versions of the TempSensor application.

The **TempSensor** application works the following way:

- The module powers on
- It connects to the latest AP it was connected to
- It starts an UDP application
- It sends the temperature recorded on the WSAB in an UDP packet send to IP 192.168.1.98, port 12345
- It suspends WiFi for 60 seconds
- It sends the new temperature recorded
- It suspends WiFi for 60 seconds
- …

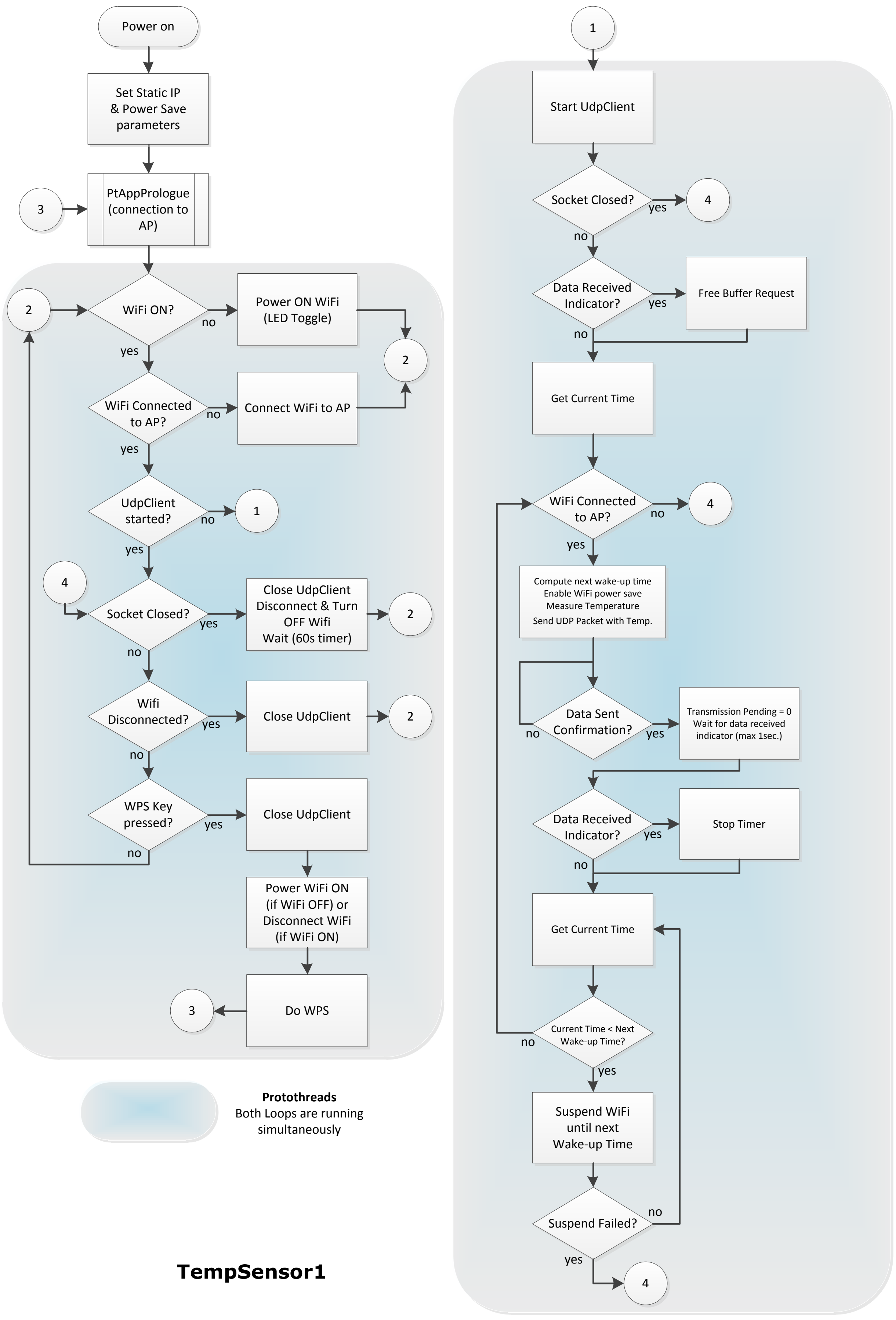
The **TempSensor2** application works the same way as TempSensor1 except for the later part:

- …
- It sends the temperature recorded on the WSAB in an UDP packet send to IP 192.168.1.98, port 12345
- It disconnect from the AP and switch OFF the WiFi module for 60 seconds
- It switch ON the WiFi and reconnect to the AP
- It sends the temperature recorded on the WSAB in an UDP packet to IP 192.168.1.98, port 12345
- …

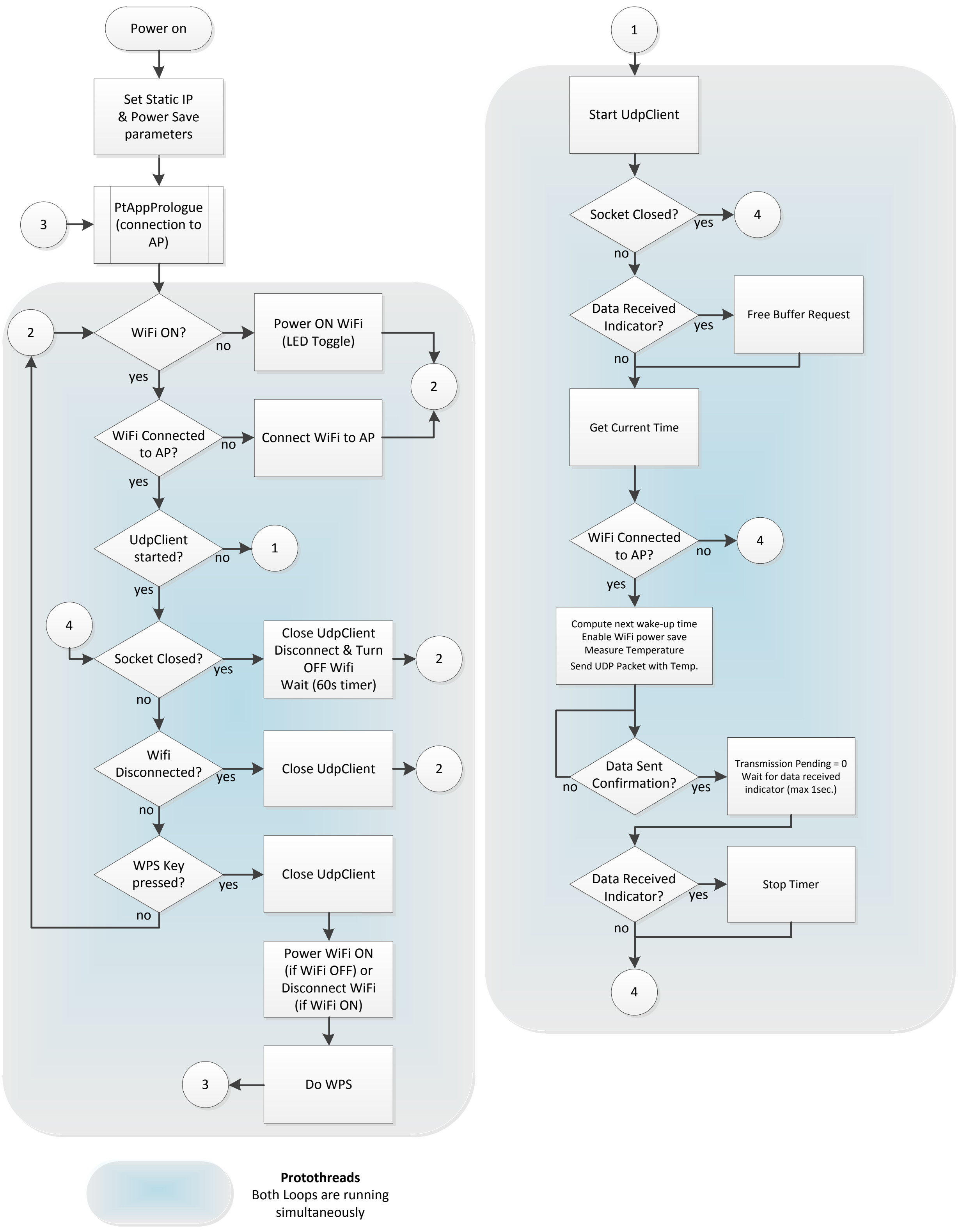
The **TempSensor3** application works the same way as TempSensor1 except for the later part:

- …
- It sends the temperature recorded on the WSAB in an UDP packet send to IP 192.168.1.98, port 12345
- It wait for 5 seconds
- It sends the temperature recorded on the WSAB in an UDP packet to IP 192.168.1.98, port 12345
- It wait for 5 seconds
- …

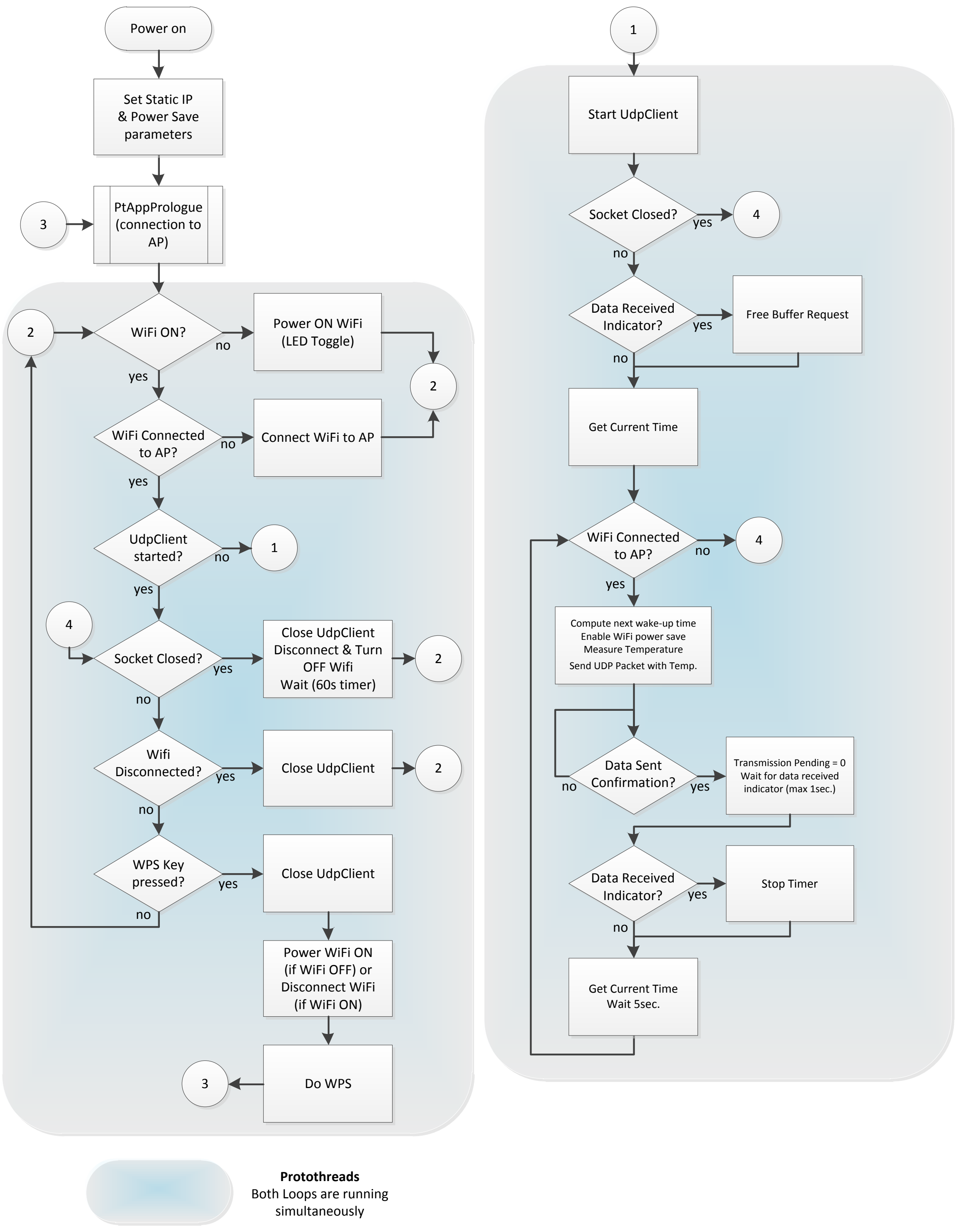
The different versions of the TempSensor application are described in the following pages.



TempSensor1


Power on
Set Static IP & Power Save parameters
PtAppPrologue (connection to AP)
3
WiFi ON?
no
yes
Power ON WiFi (LED Toggle)
2
WiFi Connected to AP?
Connect WiFi to AP
UdpClient started?
1
4
Socket Closed?
Close UdpClient Disconnect & Turn OFF Wifi Wait (60s timer)
Wifi Disconnected?
Close UdpClient
WPS Key pressed?
Power WiFi ON (if WiFi OFF) or Disconnect WiFi (if WiFi ON)
Do WPS
Start UdpClient
Data Received Indicator?
Free Buffer Request
Get Current Time
Compute next wake-up time
Enable WiFi power save
Measure Temperature
Send UDP Packet with Temp.
Data Sent Confirmation?
Transmission Pending = 0
Wait for data received indicator (max 1sec.)
Stop Timer
Protothreads
Both Loops are running simultaneously

TempSensor2



TempSensor3

5.6 The SoftAPTcpServer Application

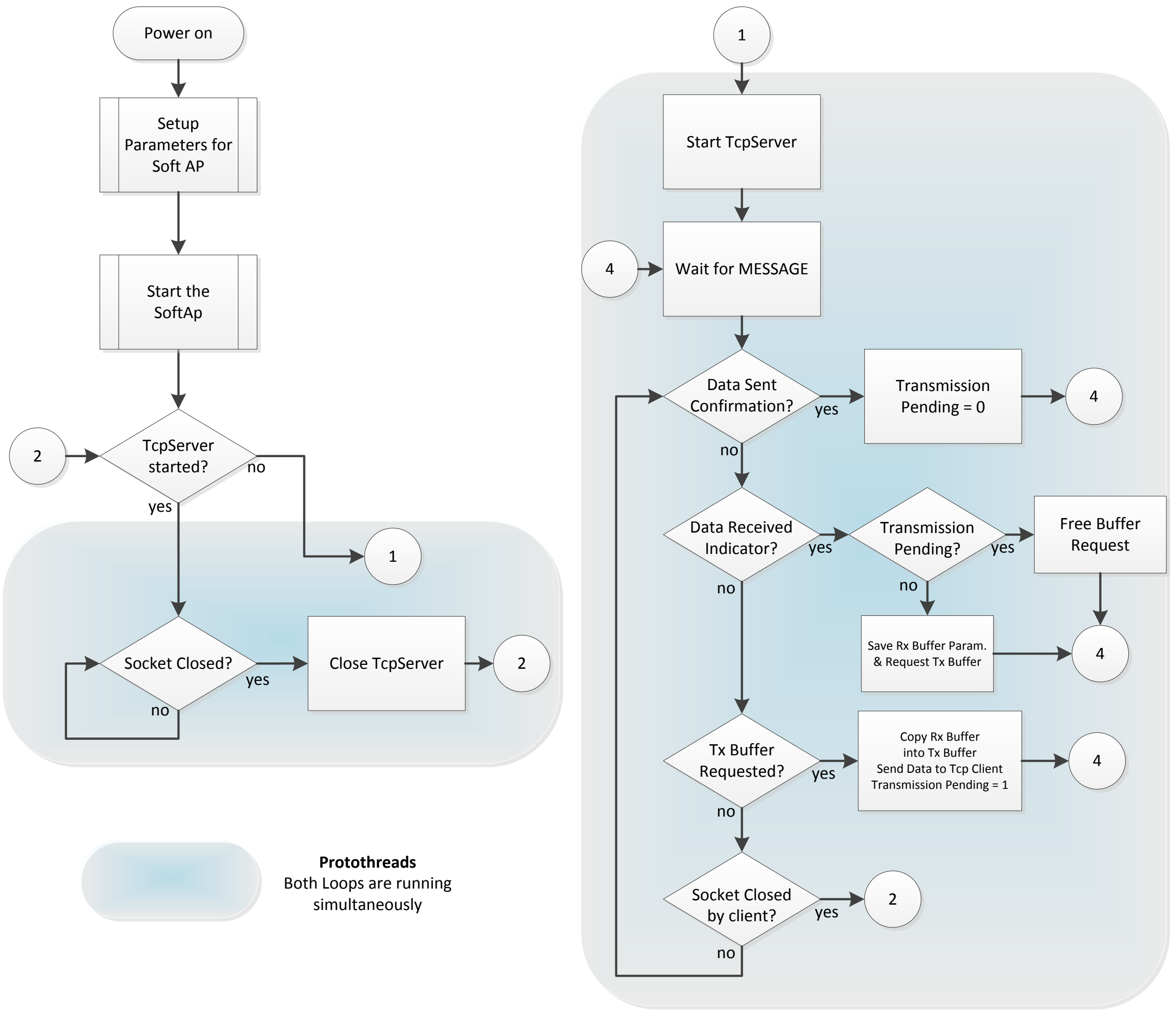
The SoftAP Tcp Server Application implements the functionalities of the SoftAP, while integrating a Tcp Server application at the same time.

The easiest way to test this application is to use two RTX4100 modules.
Load the SoftAPTcpServer application in one of them, and the Terminal application in the second module.
With the Terminal application, connect to the SoftAP using the commands:

```
> ipconfig static 192.168.1.XXX
```
(replace XXX with numbers)

```
> wifi setap Amelie_SoftAP wpa2 Takeoff_Amelie
> connect
```

Then use the command `tcp client` to the IP address 192.168.1.1, port 54321.



SoftAP TcpServer

5.7 The TempSensorSoftApCfg Application

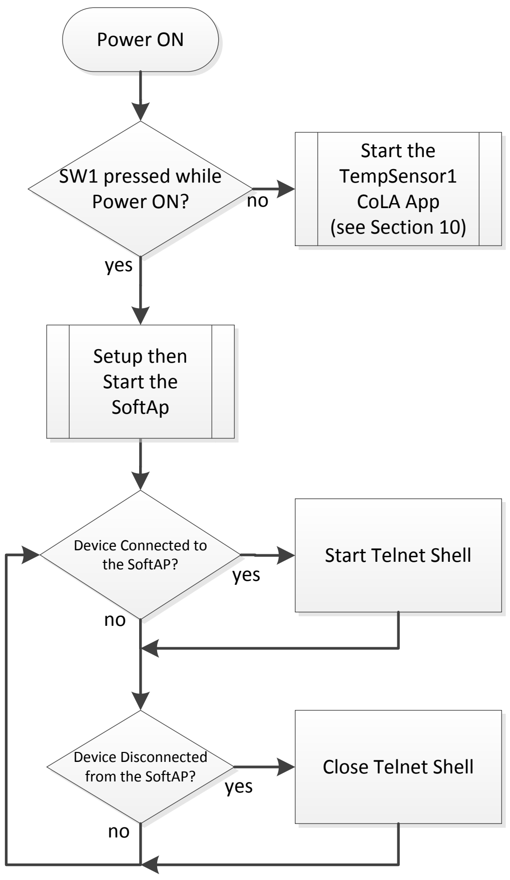
The TempSensor SoftAP Application implements the functionalities of the SoftAP, while integrating a TempSensor application at the same time.

The original purpose of this application is to demonstrate an easy procedure to associate the RTX4100 module to a customer's home Access Point.

There are 2 different options to do such an operation:

- **Option A**: The AP has a WPS button on it. After powering the module, a press on the SW2 button and the AP WPS button will associate both devices together.
- **Option B**: The AP does not have a WPS button on it. While the module is powered ON, the SW1 should remain pressed. It will power ON as a SoftAP.
  Connect to this SoftAP using another module running the Terminal application, or a Telnet Shell on a computer (after setting up a static IP).
  Once connected to the module:
  - Using Telnet, type 'o 192.168.1.1'. This will start the uart connection. Type 'help', then use the 'setap' command with the target AP parameters.
  - Using the Terminal, use the 'setap' command with the target AP parameters.

  Power cycle the module, it will connect to the target AP and start the TempSensor application.

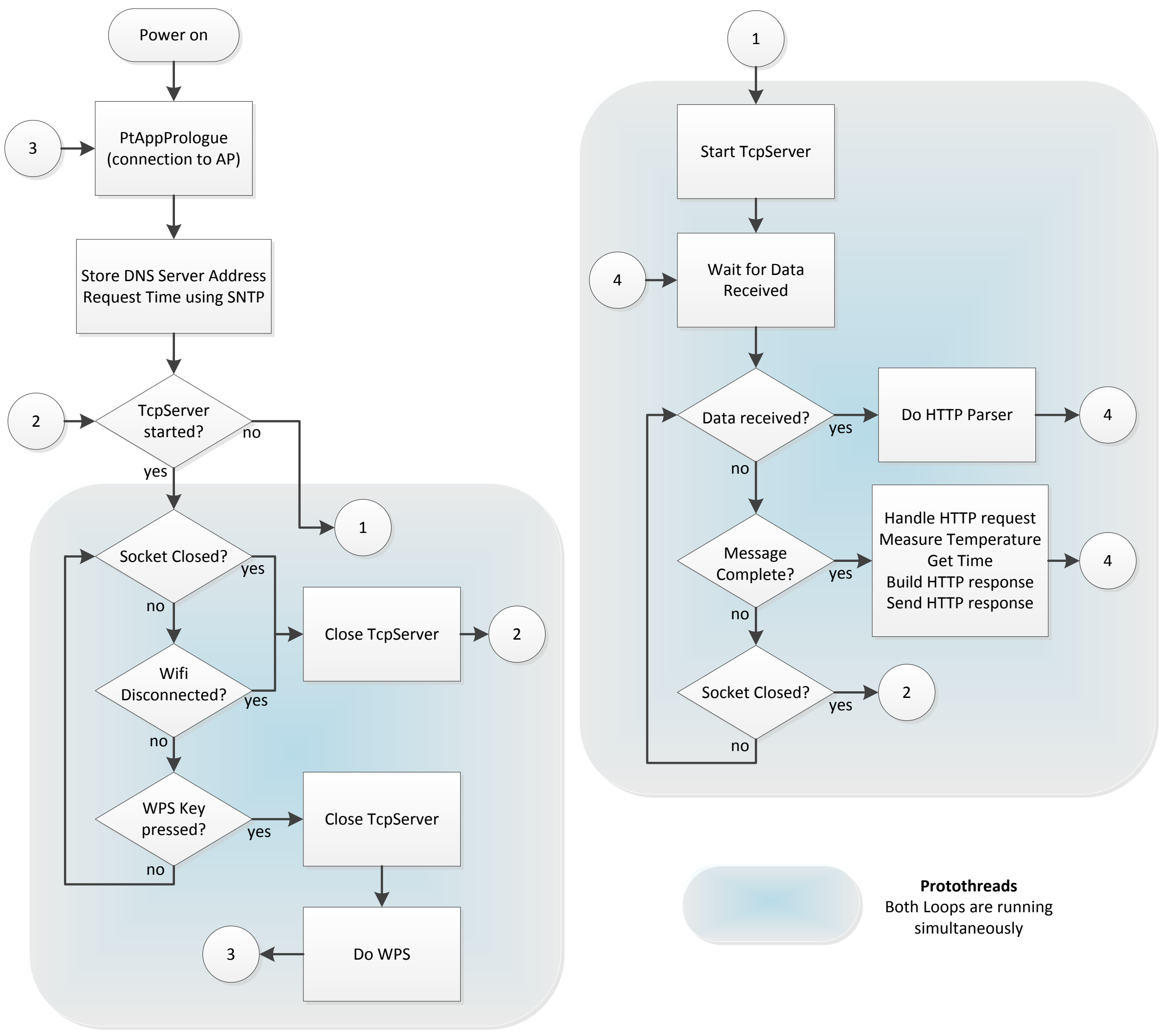

Power ON
SW1 pressed while Power ON?
no
Start the TempSensor1 CoLA App (see Section 10)
yes
Setup then Start the SoftAp
Device Connected to the SoftAP?
yes
Start Telnet Shell
no
Device Disconnected from the SoftAP?
yes
Close Telnet Shell
no

TempSensor SoftAP

5.8 WebServer Application

The Web Server Application implements the functionalities of a simple Web Server embedded on the RTX4100 module.

When the web server address is entered into a browser (default is: 192.168.1.97), the module records the temperature of the EnergyMicro chip and reports it on the webpage.



Web Server

5.9 The Tcp2Uart application

The Tcp2Uart application demonstrates a simple implementation of a data pipe. The application can run either as a TCP server or client, and will, when a connection has been established, facilitate data transmission via either LEUART1 or USART2, depending on the setting of a compiler switch. The default build uses the LEUART1.

The application can be configured by compile switches to use either one or two UARTS. The default configuration is to use the LEUART for both command mode and data interface. The data interface can be moved to USART2 by changing:

```
  #define SERIAL_PORT_LEUART 1
```

to:

```
  #define SERIAL_PORT_LEUART 0
```

in Main.c.

Both the LEUART and the USART driver supports software flow control (XON/XOFF). The driver will send XOFF to the peer device when it is unable to accept any more data (when there is less than 10 bytes free in the RX buffer) and it will send XON once the application has emptied the RX buffer. The driver will also stop the TX if an XOFF character is received from the peer device, and resume again when XON is received. The application must use the "TxStart()" functions in the driver to initiate TX in order for the driver to handle flowcontrol in the TX direction correctly.

An escape sequence is implemented in the drivers to support transmission of binary data that may include the XON(0x11) and XOFF(0x13) chars. This is done by escaping XON and XOFF chars with an escape character (0x10) which also must be escaped:

Data	Data sent
0x11	0x10 0x11
0x13	0x10 0x13
0x10	0x10 0x10

On the receiver side the escape character is removed by the driver before delivering the data to the application. The usage of software flowcontrol can be disabled by removing the following from the makefile:

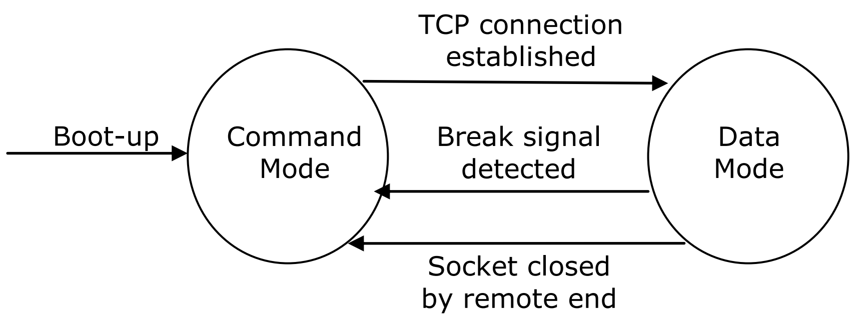
```
-DLEUART_USE_SW_FLOW_CONTROL=1 -DUSART_USE_SW_FLOW_CONTROL=1
```

There is not support for HW flow control in the source included in the SDK, but that can be implemented by the application by usage of two GPIO's to implement RS-232 CLS/RTS flowcontrol.

To run the application as a TCP server, type the following command into the application shell (using PuTTY) after connection to the AP has been established:

tcpserver <port>

Any port number can be used - 2323 is a good option. The command will make the application open the specified TCP port and wait for a client to connect. The application will enter into data mode the moment a client connects. In this mode, data received via the UART (LEUART1 or USART2) will be transmitted via TCP to the client, and data received from the client will be sent out on the UART. This relaying of data continues until either the server or client closes the socket. It is possible to close the socket and thereby revert to command mode by transmitting a Break signal through the UART. A Break signal can be sent from PuTTY, by pressing the 'Break' button on the Windows keyboard.



Running the application as a client is achieved through the following shell command:

tcpclient <server ip> <port>

The application will connect to port specified on the TCP server with the IP address specified and enter data mode once the connection with the server has been established.

Sending a Break signal via the UART, will close the socket and bring the application back in command mode in the case where a single uart is sued for both command mode and data interface.

The following commands are supported in command mode:

- version - print version information
- reboot - Reboot the system.
- erasenvs - erases all settings stored in the CoLA part of the NVS
- scan - returns available APs
- wps - initiates WPS
- ipconfig - print current IP address
- ipconfig static <static IP> [<subnet> <gateway>]
- ipconfig dhcp
- setap <ssid> none
- setap <ssid> wep <index> <key>
- setap <ssid> wpa <key>
- setap <ssid> wpa2 <key>
- getmac - get device mac address
- connect - connect to the AP
- disc - disconnect from the AP
- tcpclient <server IP> <port> - connect to TCP server
- tcpserver <port> - start a TCP server
- tcpdisc - disconnect from/stop TCP server

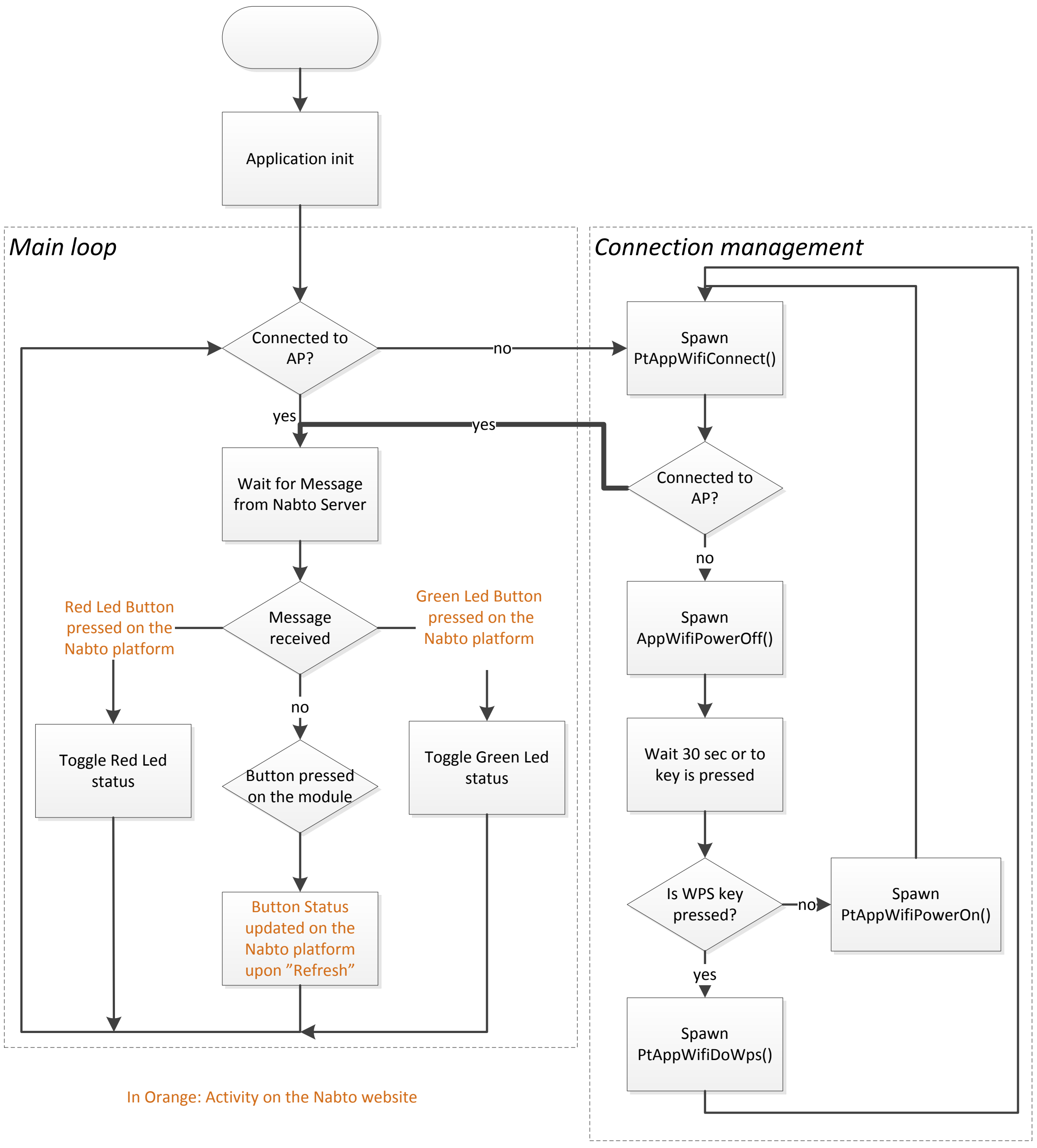
6 Cloud Services Reference Applications

RTX has developed reference applications with several cloud services partners to help you get started with your own applications using cloud services.

6.1 Nabto

The Nabto reference application works differently from the other cloud service applications in the sense that the module is not reporting data to the cloud application at regular intervals, but rather waiting for commands given by the Nabto server.
You can find a detailed guide of this specific reference application in ([QS7]).

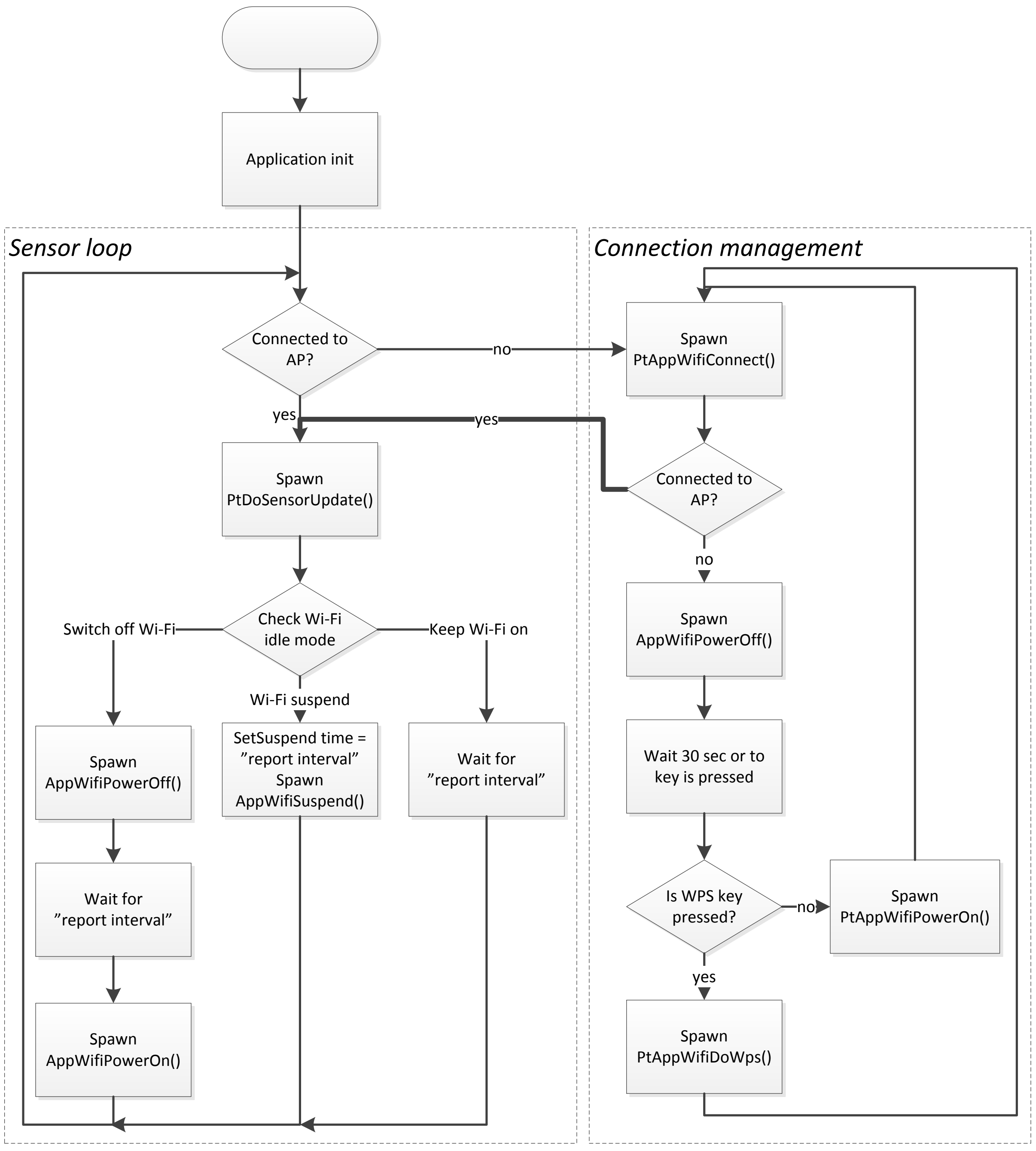
The process diagram of the application is described below:


Application init
Main loop
Connection management
Connected to AP?
no
yes
yes
Spawn PtAppWifiConnect()
Wait for Message from Nabto Server
Connected to AP?
no
Red Led Button pressed on the Nabto platform
Message received
Green Led Button pressed on the Nabto platform
Spawn AppWifiPowerOff()
no
Toggle Red Led status
Button pressed on the module
Toggle Green Led status
Wait 30 sec or to key is pressed
Button Status updated on the Nabto platform upon "Refresh"
Is WPS key pressed?
no
Spawn PtAppWifiPowerOn()
yes
Spawn PtAppWifiDoWps()
In Orange: Activity on the Nabto website

6.2 Exosite

The Exosite reference application reports the Temperature of the MCU to a Dashboard on the website www.rtxwireless.exosite.com. You can find a detailed guide of this specific reference application in ([QS6]).

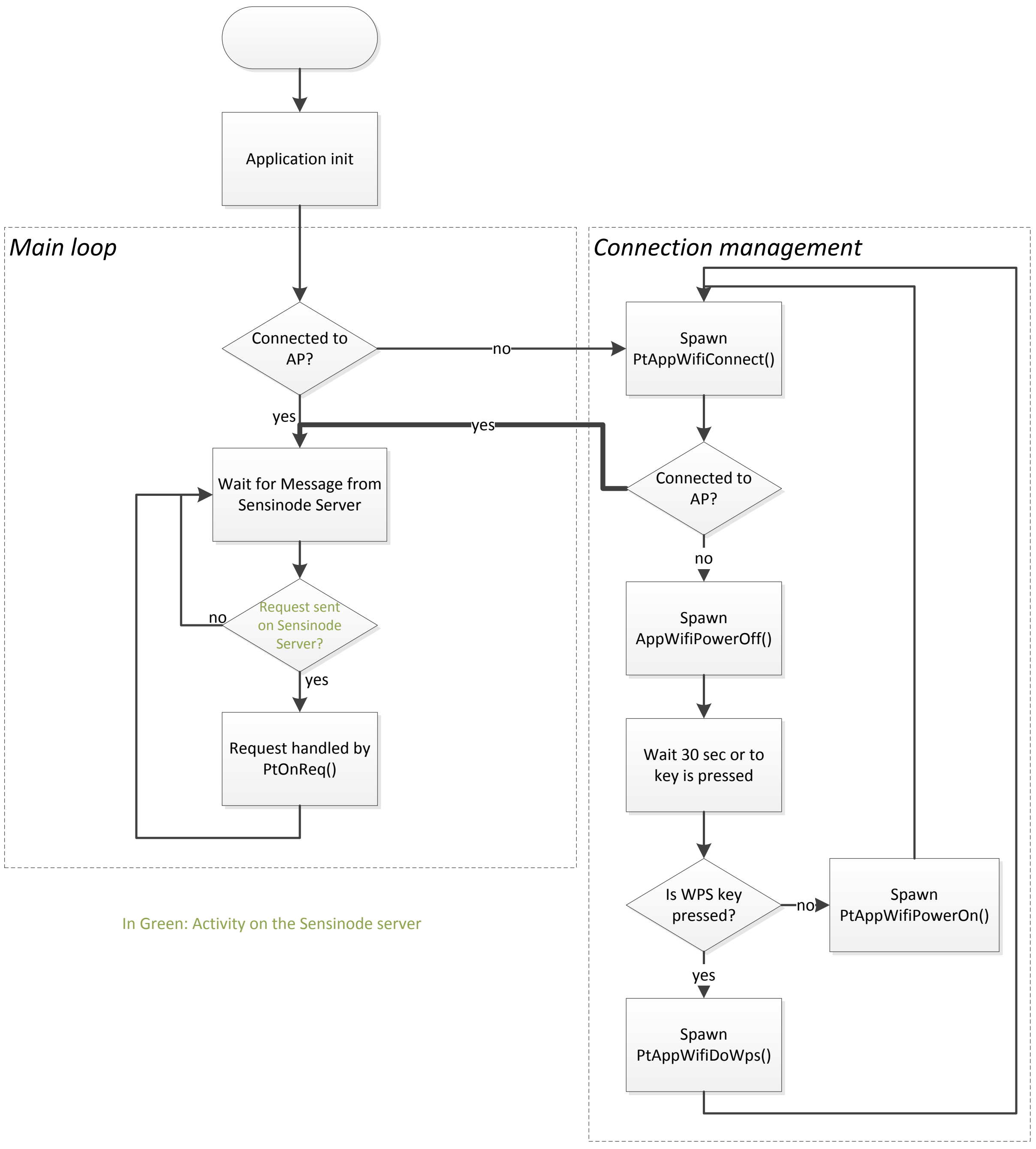
The process diagram of the application is described below:



6.3 Sensinode

The Sensinode reference application reports the Temperature of the MCU to a Floorplan on the local server http://localhost:8082. You can find a detailed guide of this specific reference application in ([QS9]).

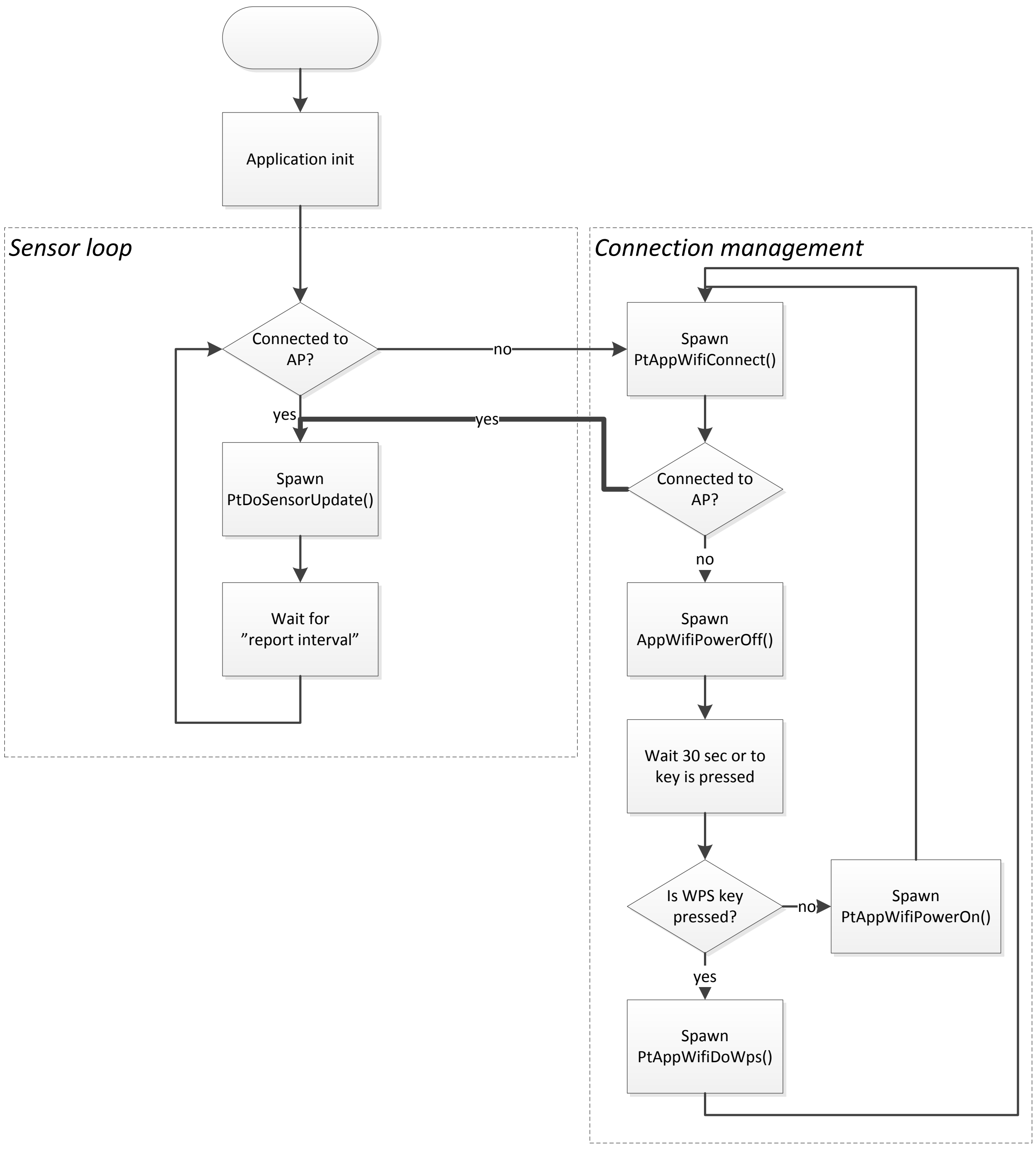
The process diagram of the application is described below:



6.4 2Lemetry

The 2Lemetry reference application reports the Temperature of the MCU and the accelerometer values to a Dashboard on the website rtx.m2m.io. You can find a detailed guide of this specific reference application in ([QS8]).

The process diagram of the application is described below:



7 Abbreviations

The following abbreviations are used in this document:

AP	**Access Point**
API	**Application Programming Interface**
BSP	**Board Support Package**
CoLA	**Co-Located Application**
RTX EAP	**RTX Embedded Access Protocol**
GPIO	**General Purpose Input/Output**
MCU	**Micro Controller Unit**
RTOS	**Real-Time Operating System**
UART	**Universal Asynchronous Receiver/Transmitter**
WEP	**Wired Equivalent Privacy**
WiFi	**Wireless Fidelity**
WPA	**WiFi Protected Access**
WPS	**WiFi Protected Setup**

8 Liability Disclaimer

General
This document and the information contained, is property of RTX A/S, Denmark. Unauthorized copying is not allowed. The information in this document is believed to be correct at the time of writing. RTX A/S reserves the right at any time to change said content, circuitry and specifications.

Information contained in this document is subject to change without notice. RTX makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. RTX shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishings, performance, or use of this material.

Warranty
This product is warranted against defects in material and Workman ship for a period of one year from date of shipment. During the warranty period, RTX will at its option, either repair or replace products, which prove to be defective. For warranty service or repair, this product must be returned to a service facility designated by RTX. Buyer shall prepay shipping charges to RTX and RTX shall pay shipping charges, duties, and taxes for products returned to RTX from another country. RTX warrants that its software and firmware designated by RTX for use with a module will execute its programming instructions when properly installed on that instrument. RTX does not warrant that the operation of the product or firmware will be uninterrupted or error free.

Limitation of Warranty
The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.
NO OTHER WARRANTY IS EXPRESSED OR IMPLIED.
RTX SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.