



ZigBee PRO Home Sensor Demo Application Note

JN-AN-1122

Revision 2.0

18-Nov-2010

Contents

About this Document	5
Organisation	5
Conventions	5
Acronyms and Abbreviations	5
Related Documents	6
Feedback Address	6
1 Demonstration Overview and Operation	7
1.1 Functional Overview	7
1.2 Setting up the Demonstration	8
1.2.1 Installing the Application Note	8
1.2.2 Loading the Demonstration	8
1.2.3 Starting the Network	9
1.3 Using the Home Sensor Demonstration	12
1.3.1 Screen Navigation and Environment Monitoring	12
1.3.2 Lighting Control	15
1.3.3 Network Control	16
1.4 Button Functions in Demonstration	17
1.4.1 Controller Board Buttons	17
1.4.2 Sensor Board Buttons	19
2 Application Design	21
2.1 Node Software Architectures	21
2.1.1 Controller Node Architecture	22
2.1.2 Sensor Node Architecture	24
2.2 Node Software Components	26
2.2.1 Controller Node [app_controller_node.c]	26
2.2.2 Buttons [app_buttons.c]	28
2.2.3 LED [app_led.c]	28
2.2.4 Display [app_display.c]	29
2.2.5 Log [app_log.c]	30
2.2.6 Start [app_start.c]	30
2.2.7 Sensor Node Components	31
2.2.8 Sensor Node [app_sensor_node.c / _SED.c]	32
2.2.9 Sample [app_sample.c / _SED.c]	33
2.2.10 Buttons [app_buttons.c]	34
2.2.11 LED [app_led.c]	34
2.2.12 Start [app_start.c (app_start_SED.c)]	35
2.2.13 System Controller [app_syscon.c]	35
3 Re-building the Application	37
3.1 Preparing for Use on High-Power Modules	37
3.2 Building and Downloading the Application	38

About this Document

This document forms part of the Application Note *ZigBee PRO Home Sensor Demo (JN-AN-1122)* which contains the source code (and associated files) of the ZigBee PRO Home Sensor Demonstration. The document describes the implementation of the demonstration application, providing operational instructions as well as an overview of the application's architecture and a description of the application code.

Organisation

This document consists of three chapters, as follows:

- [Chapter 1](#) introduces the demonstration application and describes its operation.
- [Chapter 2](#) outlines the application design, including the functions used.
- [Chapter 3](#) describes how to re-build the application and download it to the hardware.

Conventions

Files, folders, functions and parameter types are represented in **bold** type.

Function parameters are represented in *italics* type.

Code fragments are represented in the `Courier` typeface.

Acronyms and Abbreviations

API	Application Programming Interface
IDE	Integrated Development Environment
ISR	Interrupt Service Routine
JenOS	Jennic Operating System
PDM	Persistent Data Manager
PWRM	Power Manager
RTOS	Real-time Operating System
SDK	Software Developer's Kit
ZPS	ZigBee PRO Stack

Related Documents

JN-UG-3062	JN5148-EK010 Evaluation Kit User Guide
JN-UG-3048	ZigBee PRO Stack User Guide
JN-UG-3007	JN51xx Flash Programmer User Guide

Feedback Address

If you wish to comment on this document, please provide your feedback by writing to us (quoting the manual reference number and version) at the following postal address or e-mail address:

Applications
NXP Laboratories UK Ltd
Furnival Street
Sheffield S1 4QT
United Kingdom

doc@jennic.com

1 Demonstration Overview and Operation

The ZigBee PRO Home Sensor Demonstration application is intended as an aid to understanding how an application can be built on top of the ZigBee PRO stack which runs on the JN5148 device. The application can be loaded into and run on the boards of a JN5148-EK010 Evaluation Kit.



Note: For details of the evaluation kit, refer to the *JN5148-EK010 Evaluation Kit User Guide (JN-UG-3062)*. The evaluation kit boards are pre-loaded with the JenNet version of the Home Sensor Demonstration, but this can be replaced with the ZigBee PRO version (see Section 1.2).

1.1 Functional Overview

The Home Sensor Demonstration uses all five boards of the JN5148-EK010 Evaluation Kit – one Controller board and four Sensor boards - to form a system for monitoring environmental conditions in a small building, typically a house or apartment, with the boards placed in different rooms. Each board is equipped with a temperature sensor, a humidity sensor and a light-level sensor. Measurements from the Sensor boards are periodically sent to the Controller board, where they can be displayed on this board's LCD screen.

The Controller board can also display the 16-bit network address of each Sensor board and keeps a count of missed frames from each Sensor board.

In addition, a light-switch control feature is included. This allows the buttons on each Sensor board to remotely control an LED on the Controller board. Similarly, it is possible to control an LED on each Sensor board from the Controller board.

The Controller board acts as the ZigBee PRO network Co-ordinator. It is possible to control the 'Permit-Joining' state of the Co-ordinator, provided that the maximum number of Sensor boards have not joined the network. When the maximum number of Sensor boards have joined the network, Permit-Joining is automatically disabled.



Note: This limit currently only affects the Permit-Joining status on the Co-ordinator. It will still be possible for additional Sensor boards to join through Router children. However, any sensor data frames that they generate will be ignored by the Controller board (Co-ordinator).

1.2 Setting up the Demonstration

1.2.1 Installing the Application Note

The ZigBee PRO Home Sensor Demo Application Note is supplied in the ZIP file **JN-AN-1122-ZBPro-Home-Sensor-Demo.zip**. The contents of this ZIP file should be extracted into the directory **<JN5148_SDK_ROOT>\Application**, where **<JN5148_SDK_ROOT>** is the path into which the JN5148 SDK was installed (by default, this is **C:\Jennic**). The **Application** directory is automatically created when the SDK is installed.



Note: You can obtain the latest versions of the JN5148 SDK Libraries and Toolchain from www.nxp.com/jennic. The relevant part codes are JN-SW-4040 for the libraries and JN-SW-4041 for the toolchain.

1.2.2 Loading the Demonstration

In order to run the demonstration, you must first program the evaluation kit boards with the application binaries supplied in the Application Note ZIP file, as follows:

- The Controller board (with LCD screen) must be programmed as the network Co-ordinator by loading the binary file **ControllerNode_JN5148.bin** supplied in the directory **ControllerNode/Build**.
- The two Sensor boards with SMA connectors must be programmed as Routers by loading the binary file **SensorNode_JN5148.bin** supplied in the directory **SensorNode/Build**.
- The two Sensor boards with integrated PCB antennae must be programmed as End Devices by loading the binary file **SensorNode_SED_JN5148.bin** supplied in the directory **SensorNode_SED/Build**.

The binaries should be loaded into the boards using the JN51xx Flash Programmer, which is provided as part of the JN5148 SDK Toolchain (JN-SW-4041) and is described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*. This tool can be launched either directly or from within the Eclipse IDE.



Note: If you wish to use the demonstration with high-power modules, you must re-build the application binaries which are to be used on the high-power modules, as described in Chapter 3.

1.2.3 Starting the Network

Once the application binaries are loaded into the evaluation kit boards (as described in Section 1.2.2), you can assemble the demo system and run the ZigBee PRO Home Sensor Demonstration as described in the procedure below.

The precise topology of the final wireless network cannot be pre-determined since the network is formed dynamically. An example system is shown in the figure below. However, the exact system topology is not important for the operation of the demonstration.

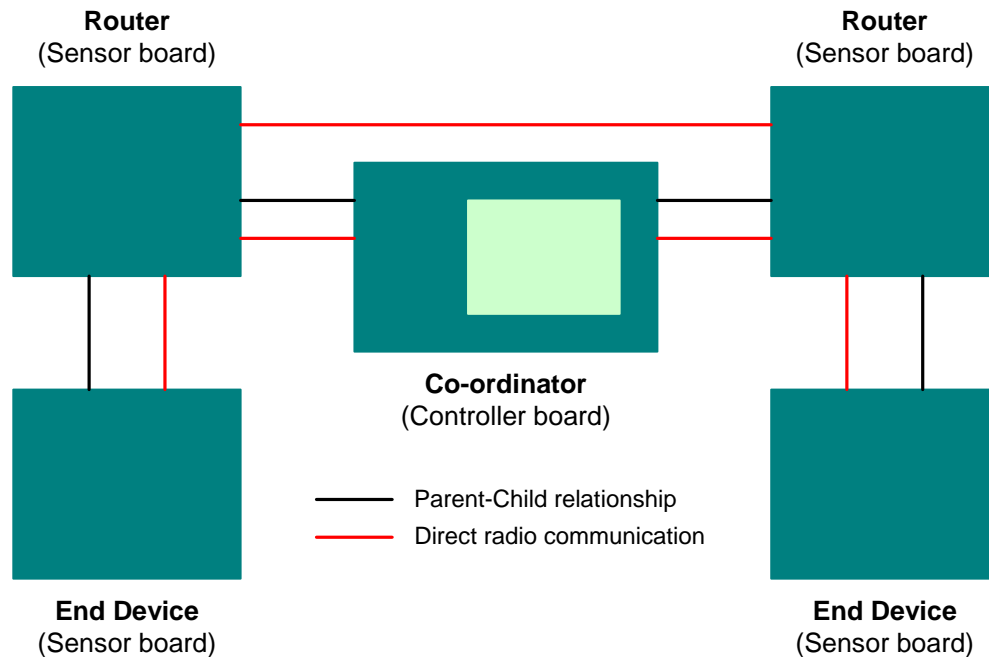


Figure 1: Example System



Note: If you have problems with a board (for example, failing to join the network), you are advised to reset or power-cycle the board and also clear the data held in its Flash memory. To do this, hold down the button SW1 and simultaneously reset/power-cycle the board - see Stack State Persistence on page 17. All the board LEDs will illuminate once the board has re-started and the Flash memory data has been erased.

Step 1 Ensure that each board is powered off but has a power source and an antenna

Check that:

- a)** all the boards are powered off (slide-switch SW6 is in the OFF position)
- b)** each board either has batteries fitted or is connected to the mains supply via an external PSU (the jumper J2 must also be in the correct position for the desired power source – for details, refer to the Reference Manual for the board type)
- c)** each board with an SMA connector has one of the supplied antennae fitted to this connector (if you need to attach an antenna, do not over-tighten - finger pressure will be sufficient)

Step 2 Power on the Controller board

Power on the Controller board using the slide-switch SW6 on the side of the board.

The power LED and the four LEDs D1-D4 will be illuminated, and the Start-up screen will appear on the LCD screen. In addition, text labels for the four buttons SW1-SW4 will appear along the lower edge of the screen.

**Step 3 Select the radio channel for the network (optional)**

If you do not want to operate the network in the default channel for the demonstration (which is 2400-MHz channel 13), you can use the two middle buttons (SW2 and SW3) on the Controller board to increase and decrease the channel number.

Step 4 Start the demonstration on the Controller board

Run the demonstration on the Controller board by pressing the button labelled 'Done' on the far right (SW4).

As the Co-ordinator, this board will create the new wireless network. As part of the network creation process, the Co-ordinator will set the channel to be used by the network according to the selection made in Step 3.

Once the network has started, the LEDs D1-D4 will be extinguished.

Step 5 Start the Sensor boards

Power on each of the Sensor boards using the slide-switch SW6 on the side of the board. You are advised to power on the Routers first (the Sensor boards with SMA-connected antennae).

As each Sensor board powers up, LED D9 on the board will be illuminated.

ZigBee PRO Home Sensor Demo

Application Note

The Routers and End Devices will then search for the network and join it. While a Sensor board is searching for the network, the board's LEDs D1 and D2 will be illuminated, but they will be extinguished once the board has joined the network (LED D2 will subsequently flash every time the board transmits sensor data). As each board joins the network, it is assigned a name which appears on the LCD screen of the Controller board - the nodes are assigned the names of rooms in the home, in the following order: Hall, Bedroom, Lounge, Bathroom. By default, the temperature measurement from each node will be displayed on the screen - this is the 'Temperature' screen of the Controller board application's user interface (see the photo below and refer to Section 1.3.1 for further details of all possible screens).



Step 6 Use the Home Sensor Demonstration

You can now use the Home Sensor Demonstration to obtain temperature, humidity and light-level measurements from the Sensor boards, as well as remotely controlling LEDs on the boards. Operation of the demo system is described in Section 1.3.



Note: If you later wish to re-run the demonstration from scratch (e.g. with a different topology), you must first clear the context data that has been automatically saved in Flash memory on the boards - see Stack State Persistence on page 17. To do this, hold down the button SW1 and simultaneously reset or power-cycle the board. All the board LEDs will illuminate once the board has re-started and the Flash memory data has been erased. You are advised to begin with the Co-ordinator (Controller board).

1.3 Using the Home Sensor Demonstration

This section describes how to navigate around the screens of the Home Sensor Demonstration on the Controller board and how to use the functionality of the system.

The Home Sensor Demonstration has three sets of functionality:

- Environment monitoring (temperature, humidity, light-level) - see Section 1.3.1
- Lighting control - see Section 1.3.2
- Network control - see Section 1.3.3

Screen navigation is also explained in Section 1.3.1.

1.3.1 Screen Navigation and Environment Monitoring

The sensor readings from the Sensor boards of the demo system are periodically sent to the Controller board where they can be displayed on the LCD screen. The application which runs on the Controller board provides a user interface consisting of a set of information screens that display the sensor data in different ways. This section describes these screens and how to use the four buttons SW1-SW4 to navigate around them. This navigation is illustrated in Figure 2 below and the screens are then described.



Note 1: Once the demo system is up and running as described in Section 1.2.3, you will have reached the 'Temperature' screen.



Note 2: The functions of the buttons SW1-SW4 for the various screens are summarised in Section 1.4.

ZigBee PRO Home Sensor Demo Application Note

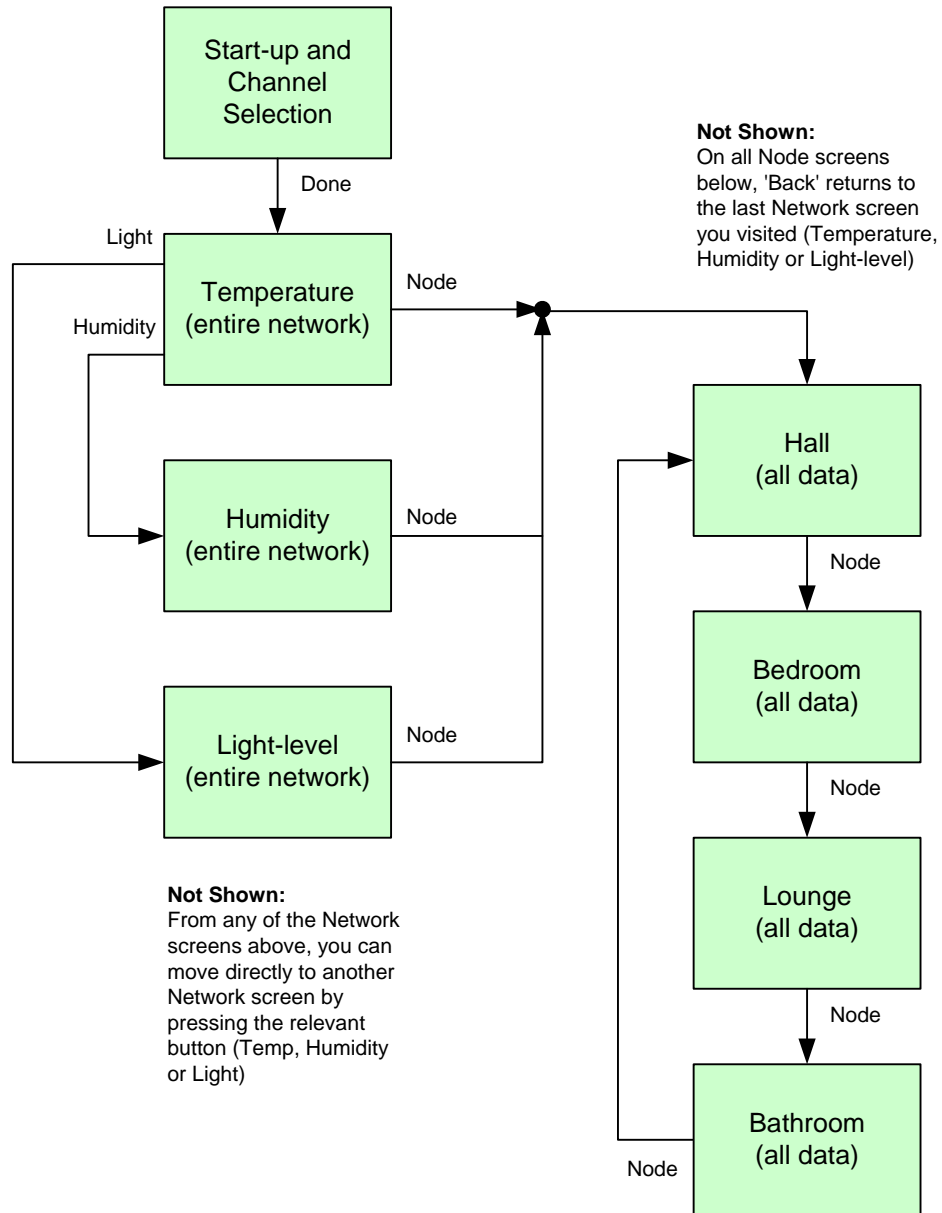


Figure 2: Screen Navigation on Controller Board

As seen in Figure 2, after the Start-up screen, the information screens are divided into two groups, Network screens and Node screens, described below.

Network Screens

Each Network screen displays a particular type of sensor data for all network nodes:

- **Temperature screen:** This screen displays the temperature readings from all nodes, in degrees Celsius. It is the first screen displayed following the Start-up screen when the network starts (once the channel has been selected and the 'Done' button has been pressed on the Start-up screen).
- **Humidity screen:** This screen displays the relative humidity readings from all nodes, as percentages.
- **Light-level screen:** This screen displays the light-level readings from all nodes. A light-level reading is represented pictorially by a circle which is partially filled according to the light-level - a completely filled circle indicates total darkness and a completely unfilled circle indicates maximum brightness.

Each sensor reading is also displayed as a graph as well as a value/picture.

The network address of each node is also shown as a 4-digit hexadecimal number. In addition, the 'Permit-Joining' status of the Co-ordinator is displayed - see Section 1.3.3.

An example of the 'Humidity' screen is shown in Figure 3 below.

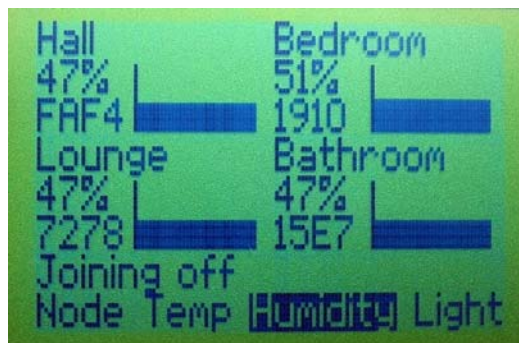


Figure 3: Example Network Screen (Humidity)

You can move directly from one Network screen to another Network screen by pressing the appropriate button on the Controller board (Temp, Humidity or Light). You can also move to the first Node screen (see below) by pressing the 'Node' button.

Node Screens

Each Node screen displays all the sensor readings from a particular node (Hall, Bedroom, Lounge or Bathroom). The Node screens are accessed by pressing the 'Node' button on any of the Network screens. This takes you to the first Node screen, which is 'Hall'. Pressing the 'Node' button repeatedly then takes you through the Node screens in rotation: first to the 'Bedroom' screen, then to the 'Lounge' screen, followed by the 'Bathroom' screen and then back to the 'Hall' screen.

The Node screens also display the number of messages from the relevant node that have been missed - this count will increase rapidly if the network has lost the node.

You can exit the Node screens by pressing the 'Back' button, which will take you back to the last Network screen that you visited.

An example of the 'Hall' screen is shown in Figure 4 below.

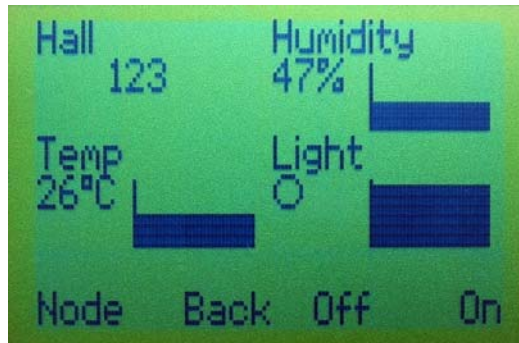


Figure 4: Example Node Screen (Hall)

The 'Off' and 'On' buttons are used to control LEDs, as described in Section 1.3.2.

1.3.2 Lighting Control

The Home Sensor Demonstration illustrates wireless lighting control by using a button on one board to control an LED on another board, thereby providing a wireless light-switch. Lighting control is implemented in two ways:

- Buttons on a Sensor board are used to control an LED on the Controller board.
- Buttons on the Controller board are used to control an LED on a nominated Sensor board.

Controlling an LED on the Controller Board

Buttons SW1 and SW2 on any Sensor board can be used to control one of the four LEDs on the Controller board. The LEDs controlled from the different Sensor boards are detailed in the table below.

Sensor Board	LED on Controller
Hall	D1
Bedroom	D2
Lounge	D3
Bathroom	D4

The buttons SW1 and SW2 on the Sensor boards are used as follows:

- Pressing SW2 illuminates the relevant LED on the Controller board.
- Pressing SW1 extinguishes the relevant LED on the Controller board.

There is no visible effect on the Sensor board itself.

Controlling an LED on a Sensor Board

Once you have navigated on the Controller board to the screen for a particular node (e.g. Lounge), as described in Section 1.3.1, you can use the buttons SW3 and SW4 on the Controller board to control LED D1 on the remote Sensor node:

- Pressing SW4 ('On') illuminates the LED.
- Pressing SW3 ('Off') extinguishes the LED.

There is no visible effect on the Controller board or on any of the other Sensor boards.

1.3.3 Network Control

Two aspects of network functionality can be controlled using the on-board buttons:

- 'Permit-Joining' on the Co-ordinator (Controller board)
- 'Stack state persistence' on any of the nodes

'Permit-Joining' on Co-ordinator

The Co-ordinator (Controller board) allows other nodes to join the network through it, by default. However, this 'Permit-Joining' state can be disabled, in which case the Co-ordinator will no longer allow other nodes to join the network through it (although they will still be able to join via the Router nodes, if there are any Routers in the network).

The ability to disable and re-enable Permit-Joining on the Co-ordinator is provided through the button SW1 on the (Controller) board. To use this functionality, you must be on one of the Network screens (Temperature, Humidity or Light-level). Pressing and holding down button SW1 for 2 seconds will then toggle the Permit-Joining function - the current setting is shown on the LCD screen as 'Joining on' or 'Joining off'.

You can use this feature when starting the network, to help force the shape of the network. For example, to avoid a 'Star' network topology, disable Permit-Joining on the Co-ordinator immediately after the first Router has joined the network - subsequently, joining nodes will be forced to join the network through the Routers.

If the network attains its maximum number of Sensor boards (4 in this demonstration), Permit-Joining on the Co-ordinator is automatically disabled and becomes inaccessible through SW1.



Note: The Permit-Joining status is ignored for a re-join. Therefore, if a node leaves the network (e.g. as the result of power-cycling the board) and then attempts to re-join via the Co-ordinator, the Permit-Joining setting on the Co-ordinator is bypassed and the re-join is always allowed.

Stack State Persistence

In this demonstration, the state of the ZigBee PRO stack on a node is maintained through node reset and power-cycle events. This is done by saving stack and application context data in non-volatile memory, so that this data can be recovered by the JN5148 device after a reset or power-cycle. Therefore, following one of these events, the node will remember and re-take its previous place in the network. You can, however, make the node forget its previous network status by erasing the contents of non-volatile memory during a reset or power-cycle. To do this, press any button on the board (one of SW1-SW4 for the Controller board or SW1-SW2 for a Sensor board) while pressing the RESET button or power cycling using the ON-OFF switch (SW6). Once the stored context data has been erased, all the board LEDs will illuminate (the LEDs will extinguish when the board joins another network and context data is saved).

1.4 Button Functions in Demonstration

This section summarises the functions of the buttons on the Controller board (buttons SW1-SW4) and Sensor boards (buttons SW1-SW2) in the ZigBee PRO Home Sensor Demonstration.



Note: Holding down any one of these buttons while resetting or power-cycling the board will cause any stack and application context data saved in non-volatile memory to be erased - see Section 1.3.1.

1.4.1 Controller Board Buttons

The tables below summarise the functions of the four buttons SW1-SW4 on the Controller board in the demonstration.

Start-up Screen

Button	Label	Function
SW1	Ch 13 (initial label)	Button has no function, but label displays currently selected channel (which is changed using SW1 and SW2 - see below).
SW2	+	Increment channel number (within range 11-26).
SW3	-	Decrement channel number (within range 11-26).
SW4	Done	Start network (to operate in the selected channel).

Table 1: Button Functions for Start-up Screen

Node Screens (Hall, Bedroom, Lounge, Bathroom)

Button	Label	Function
SW1	Node	Go to next Node screen in the cycle: Hall→Bedroom→Lounge→Bathroom→Hall...
SW2	Back	Go back to last Network screen visited.
SW3	Off	Extinguish LED D1 on the currently displayed node (Sensor board).
SW4	On	Illuminate LED D1 on the currently displayed node (Sensor board).

Table 2: Button Functions for Node Screens**Network Screens (Temperature, Humidity, Light-level)**

Button	Label	Function
SW1	Node	Go to first Node screen (for 'Hall'), which shows all the sensor readings for that node. Holding down this button for 2 seconds toggles the 'permit joining' state of the Co-ordinator.
SW2	Temp*	Go to Temperature screen, which shows temperature readings (°C) for all nodes.
SW3	Humidity*	Go to Humidity screen, which shows relative humidity readings (%) for all nodes.
SW4	Light*	Go to Light-level screen, which shows light-level readings for all nodes.

Table 3: Button Functions for Network Screens

* If label is inverted (dark background), this means you are on the corresponding Network screen.

1.4.2 Sensor Board Buttons

The table below summarises the functions of the two buttons SW1 and SW2 on the Sensor boards in the demonstration. These buttons are used to remotely control an LED on the Controller board, the exact LED depending on the Sensor node (Hall, Bedroom, Lounge or Bathroom).

Button	Sensor Node	Function
SW1	Hall	Extinguish LED D1 on the Controller board.
	Bedroom	Extinguish LED D2 on the Controller board.
	Lounge	Extinguish LED D3 on the Controller board.
	Bathroom	Extinguish LED D4 on the Controller board.
SW2	Hall	Illuminate LED D1 on the Controller board.
	Bedroom	Illuminate LED D2 on the Controller board.
	Lounge	Illuminate LED D3 on the Controller board.
	Bathroom	Illuminate LED D4 on the Controller board.

Table 4: Button Functions for Remote Control of LEDs D1-D4

2 Application Design

This chapter describes the application code that runs on the different device types in the ZigBee PRO Home Sensor Demonstration.



Note 1: In the ZigBee PRO Libraries (JN-SW-4040) release v1.3 and above, the network joining procedure in busy RF locations should be handled within the application, in order to minimise the required stack resources. The steps to achieve this are illustrated in the code for this Application Note and described in the *SDK Libraries Installer v1.3 Release Notes (JN-RN-0023)*.



Note 2: By setting the value of the network parameter *apsUseExtendedPanId* to zero in the configuration file (using the ZPS Configuration Editor), the Sensor nodes are configured to try to join the first suitable network that they discover. However, if this parameter is set to a non-zero value, the Sensor nodes will try to join the network with the Extended PAN ID (EPID) specified by *apsUseExtendedPanId*.

2.1 Node Software Architectures

Descriptions follow for the Controller and Sensor nodes. Note that there are two types of Sensor node – Router and Sleeping End Device (SED). These Sensor node types are similar but, where necessary, additional notes detail the differences between them.

2.1.1 Controller Node Architecture

The architecture of the Controller node application is shown in Figure 5 below.

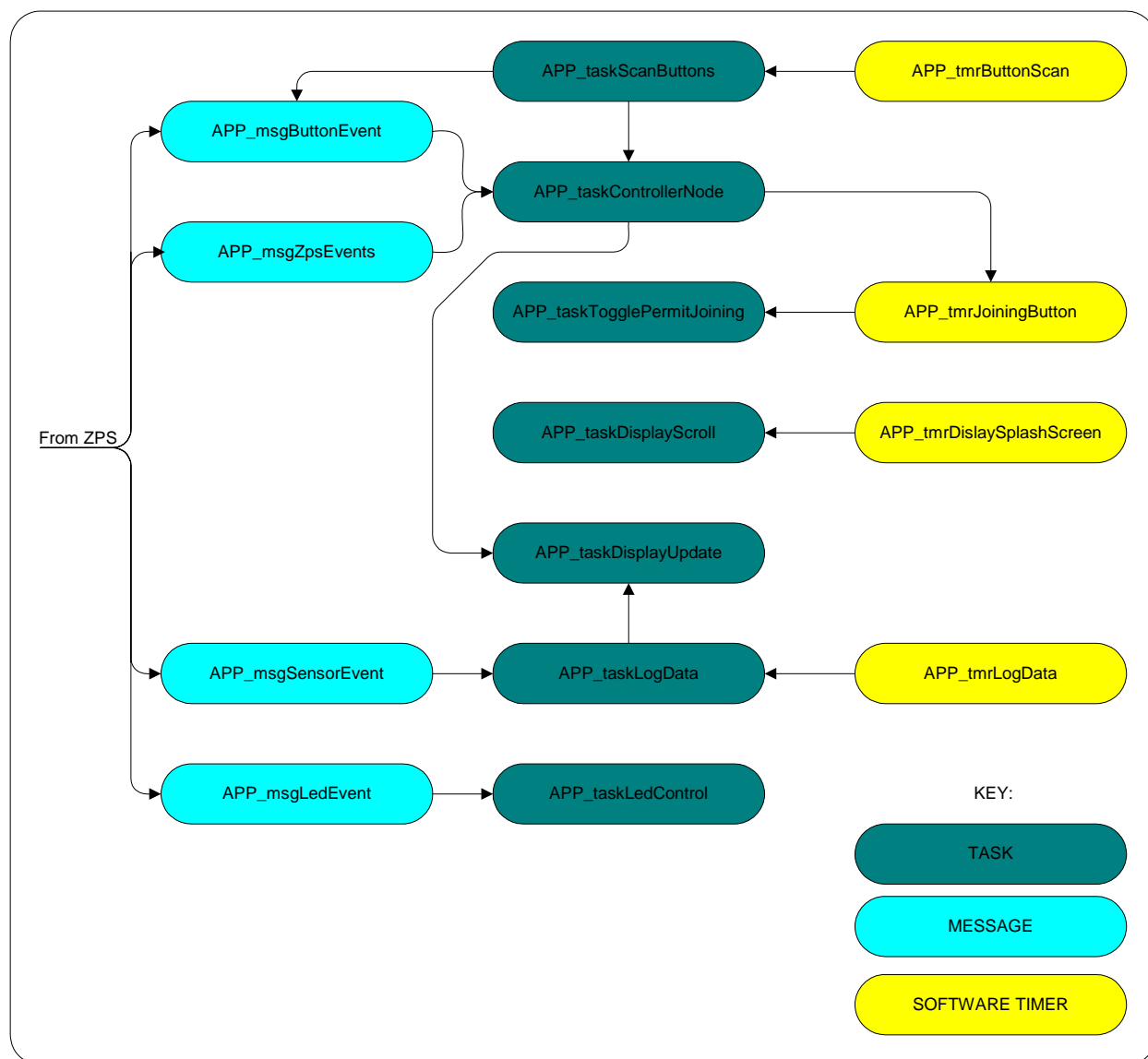


Figure 5: Architecture of Controller Node Application

The Controller node application is implemented in the following tasks:

APP_taskControllerNode

This is the Controller node application's main task. It is activated by stack events generated by the ZigBee PRO Stack (ZPS), which posts messages containing the details of the stack event to the ControllerNode task. It is also activated by messages posted from the buttons task. The ControllerNode task contains the state machine necessary to start a network and subsequently function as the application controller, interpreting button presses and controlling which sensor data is shown on

the LCD screen. This task is also responsible for sending LED_CONTROL frames to the Sensor nodes in order to switch the remote LEDs on or off, if the user presses the appropriate buttons.

APP_taskScanButtons

This task is activated by a software timer, which is itself activated by the System Controller ISR (Interrupt Service Routine) whenever a button interrupt is generated. This task de-bounces the buttons. When a button press is successfully de-bounced, the task generates a BUTTON_UP or BUTTON_DOWN event, and posts a message to the main Controller task containing details of the button event.

APP_taskLogData

The LogData task implements the SENSOR endpoint, which receives data frames, containing the sensor measurement data, from the Sensor nodes. When a Sensor node first sends data to the Controller, the node's address is registered by the LogData task and the sensor data is added to a log. The task stores a rolling record of the last 16 data sets sent by the Sensor node. The task is activated by the ZPS task, which posts messages containing the details of the data frame. It is also periodically activated by a software timer, which causes the task to advance the rolling data log by one interval.

APP_taskDisplayScroll

This task is activated by a software timer, and is used to generate a scrolling title display on the initial splash screen.

APP_taskDisplayUpdate

The DisplayUpdate task is activated directly by other tasks. The main Controller task activates it when the display mode has been changed in response to a button press, and the LogData task activates it when the sensor data has been updated. The task rebuilds the display using the updated display mode or sensor data.

APP_taskLedControl

This task implements the LED_CONTROL endpoint. It receives LED control frames from the Sensor nodes, and switches the Controller's LEDs on or off accordingly. The task is activated by the ZPS task, which posts messages containing the details of the data frame. It also receives any confirmations generated when other tasks send LED Control frames to the sensor nodes, as these outgoing frames are also sent through the LED_CONTROL endpoint.

APP_taskTogglePermitJoining

This task is activated by a software timer, which is started running when the user requests the Controller to toggle its Permit-Joining state. The actual functionality is that Permit-Joining is toggled if button SW1 is pressed for more than two seconds (when the "Network" display mode is active). The button press starts the software timer. On expiry, it activates the TogglePermitJoining task. If the button is released before the 2 seconds have passed, the main Controller task stops the timer before it expires, preventing this task from being called.



Note: The software timers use the tick-timer. The necessary callbacks are implemented in the common files **app_timer_driver.c** and **app_timer_driver.h**. For further information on the JenOS software timers, refer to the *ZigBee PRO Stack User Guide (JN-UG-3048)*.

2.1.2 Sensor Node Architecture

The architecture of the Sensor node application is shown in Figure 6 below:

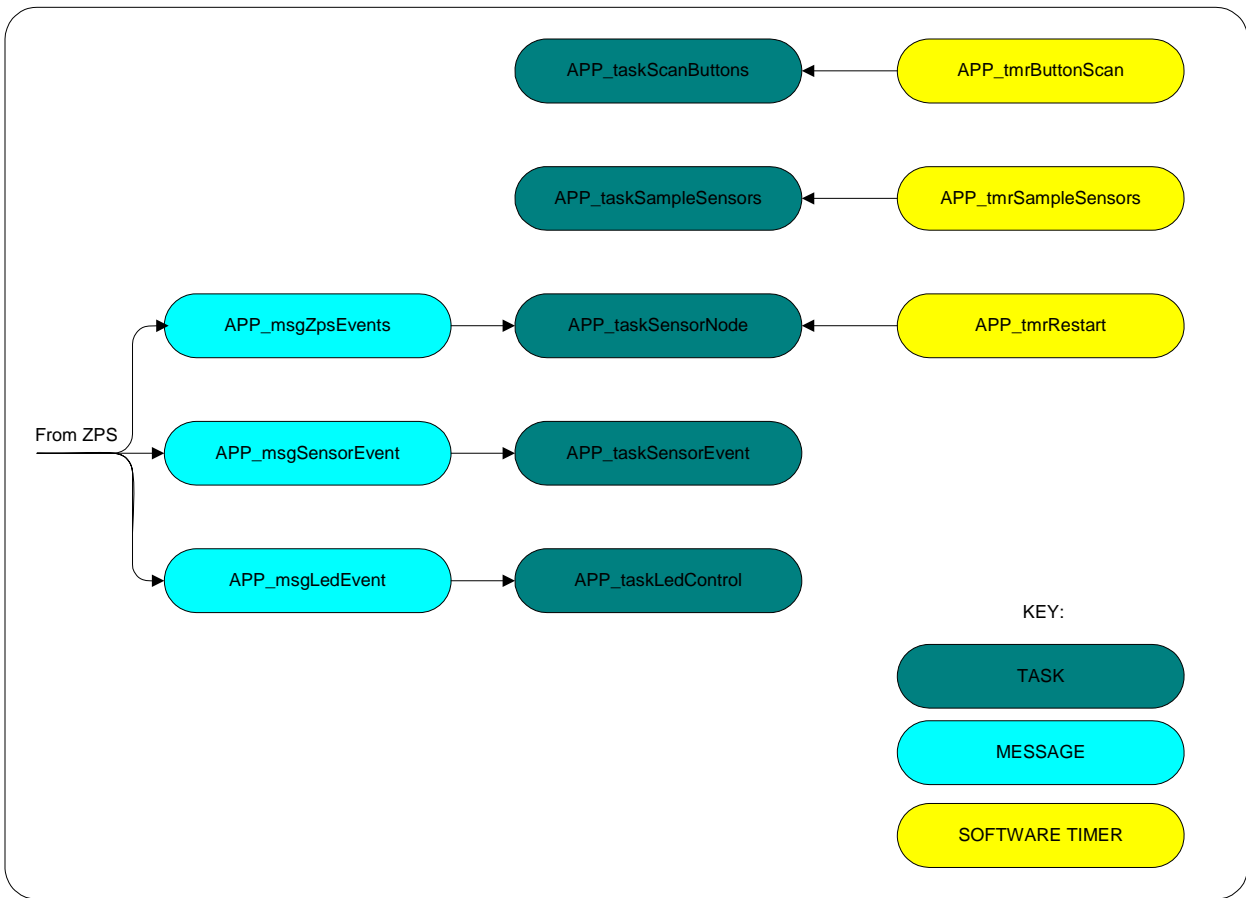


Figure 6: Architecture of Sensor Node Application

The Sensor node application is implemented in the following tasks:

APP_taskSensorNode

This is the Sensor node application's main task. It is activated by stack events generated by the ZPS, which posts messages containing the details of the stack event to the SensorNode task. The SensorNode task contains the state machine which is necessary to discover and join an appropriate network.

APP_taskSampleSensors

The SampleSensors task contains a state machine which co-ordinates the taking of measurements from the node's light, temperature and humidity sensors. Upon completion of the sensor readings, the task sends a SENSOR_DATA frame containing the readings to the Controller node. If the Sensor node is a Router, this task is periodically activated by a software timer. If the Sensor node is a Sleeping End Device, the task is activated on wake-up from within the Wakeup callback.

APP_taskScanButtons

This task is activated by a software timer, which is itself activated by the System Controller ISR whenever a button interrupt is generated. This task de-bounces the buttons. When a button press is successfully de-bounced, the task will send an LED_CONTROL frame to the Controller node, switching the remote LED on or off.

APP_taskLedControl

This task implements the LED_CONTROL endpoint. It receives LED control frames from the Controller node, and switches an LED on the Sensor board on or off accordingly. The task is activated by the ZPS task, which posts messages containing the details of the data frame. It also receives any confirmations generated when other tasks send LED Control frames to the Controller node, as these outgoing frames are also sent through the LED_CONTROL endpoint.

APP_taskSensorEvent

This is an additional, and trivial, task which exists on Router-Sensor nodes, but not on SED-Sensor nodes. Both Sensor node types illuminate LED D2 to indicate activity, but the behaviour is implemented slightly differently on the two node types:

- On SED-Sensor nodes, the LED is illuminated when the device wakes and is then extinguished when the device goes to sleep.
- On Routers-Sensor nodes, the LED is illuminated when a sensor data frame is passed to the stack for transmission and is then extinguished when the stack passes a Confirmation back to the application.

This task implements the SENSOR_DATA endpoint and therefore receives the APS Confirm event generated by the ZPS after the SENSOR_DATA frame has been sent. The ZPS posts a message containing the Confirm, and the message activates the task. The task then switches LED D2 off. It is easy to distinguish a Router from a SED, as LED D2 flashes briefly on a Router but illuminates more solidly on a SED.



Note: The software timers use the tick-timer. The necessary callbacks are implemented in the common files **app_timer_driver.c** and **app_timer_driver.h**. For further information on the JenOS software timers, refer to the *ZigBee PRO Stack User Guide (JN-UG-3048)*.

2.2 Node Software Components

The Controller node consists of a number of components which correspond to individual source files. These implement the tasks and ISRs mentioned above in Controller Node Architecture, contain the various data structures along with access functions where appropriate, and additional low-level functions such as start-up routines and power management callbacks.

Table 5 below lists the components and corresponding filenames, and briefly describes their purpose.

Component	File (.c/.h)	Purpose
Controller Node	app_controller_node	Contains the main ControllerNode task, along with functions necessary to manage network formation and handling of button events. Responsible for initialisation of the ZigBee PRO stack and the application. Generates LED_CONTROL frames.
Buttons	app_buttons	Contains the ScanButtons task, which debounces button presses and posts appropriate messages to the ControllerNode task.
LED	app_led	Contains the LedControl task, which receives LED_CONTROL frames, and switches LEDs on and off.
Display	app_display	Contains the DisplayUpdate and DisplayScroll tasks, along with the Display state variable, a number of access functions and various display maintenance functions that are called by the two display tasks.
Log	app_log	Contains the Sensor Log data structure, along with the LogData task and a number of access functions.
Start	app_start	Contains the entry point through which the device starts running after a reset or power-up, and is responsible for starting the JenOS modules including the RTOS. Also contains some exception handlers and callbacks for the Power Manager.

Table 5: Controller Node Components

The Controller node components are described in detail in the sub-sections below.

2.2.1 Controller Node [app_controller_node.c]

This component maintains a data structure that is responsible for the node's operational state, its Permit-Joining state, and the radio channel that it is operating on. This data is registered with the JenOS Persistent Data Manager (PDM), and therefore persists through resets and power-cycles.

It contains the following tasks and functions:

APP_vInitialiseControllerNode()

This function is called during start-up. It initialises the hardware, the ZigBee PRO stack and the application. As part of this initialisation, the initial states of the buttons are checked. If any buttons are pressed at start-up, a call is made to the PDM requesting that the entire ZigBee PRO stack and application context is erased from Flash memory, thus causing a clean start. Otherwise, the function checks if a valid record has been retrieved from Flash memory by the PDM. If this is the case, the function re-starts the ZigBee PRO stack and application - the network already exists and does not need to be formed. If no record is retrieved, the function starts the ZigBee PRO stack and application from fresh, resulting in the formation of a new network.

OS_TASK(APP_taskControllerNode)

This is the application's main task. It collects stack event messages and application messages, and then, depending on the application's state, calls one of a number of lower-level functions to handle the event.

vHandleStartupEvent()

This is the initial state. The function displays the start-up screen before advancing to the next state.

vHandleConfigureNetworkEvent()

This function changes the radio channel in response to button presses, then starts the ZigBee PRO stack.

vHandleNetworkFormationEvent()

This function waits for the network to indicate that it has successfully formed, then sets Permit-Joining to the ON state. It advances the display state machine to the Network screen, starts the logging module and switches on the LEDs to indicate that the network has formed. It advances the application state machine to the next state and then calls the PDM, requesting it to save the ControllerNode's data record to Flash memory.

vHandleNetworkScreenEvent()

This function is called when the Network screen mode is active. It responds to button presses and may start the Permit-Joining toggle timer running, or select which of the three sensor types (temperature, humidity, light) is being displayed for the current node. It may also request a change of display from "Network" to "Node".

vHandleNodeScreenEvent()

This function is called when the Node screen mode is active. It responds to button presses and may request the Display component to change the Sensor node for which sensor data is currently being shown. It may also change the display or initiate sending of an LED_CONTROL frame.

OS_TASK(APP_taskTogglePermitJoining)

If the maximum number of children (Sensor nodes) has not been reached, this task will toggle the Permit-Joining state and then update the display.

vCheckStackEvent()

This function handles stack events when the network has formed. For most events, there is no action required.

vHandleLedControlEvent()**vSendLedData()**

Between them, these two functions form a data frame containing LED_CONTROL data and then pass the frame to the ZigBee PRO stack for transmission to a Sensor node.

2.2.2 Buttons [app_buttons.c]

APP_bButtonInitialise()

This function initialises the DIO lines that the buttons use, setting up levels and enabling interrupts appropriately.

OS_TASK(APP_taskScanButtons)

This task debounces button presses. It is activated 8 times by a software timer per button press. After the final call, if the state of the button was constant each time the task was activated then a button event is generated. A message is posted to the ControllerNode task with the details of the button event.

2.2.3 LED [app_led.c]

APP_vLedsInitialise()

This function initialises the LEDs, ensures that they are off, and disables the UART1's CTS and RTS lines, which share DIO lines with the LEDs.

APP_vLedSet()

This function allows other components to switch LEDs on and off. This is used, for example, by the Controller Node task to switch all the LEDs off when the network has been formed.

OS_TASK(APP_taskLedControl)

This task receives messages associated with the LED_CONTROL endpoint. These contain frames from the Sensor nodes and, according to their content, switch the corresponding LED (D1-D4) on or off.

2.2.4 Display [app_display.c]

This component maintains data objects that store the current display state and the currently selected Sensor node.

APP_vDisplayInitialise()

This function resets the LCD screen.

APP_u8GetCurrentNode()

APP_vDisplayCycleNode()

These two functions provide access to the current selected node object.

APP_vDisplayUpdate()

This function allows other components to request that the display is rebuilt. This will normally be called after the display mode has been changed or the sensor data has been updated.

OS_TASK(APP_taskDisplayUpdate)

This task checks the current display mode and then calls the appropriate lower level functions to rebuild the display.

vBuildSplashScreen()

vUpdateSplashScreen()

vBuildNetworkScreen()

vUpdateNetworkScreen()

vBuildNodeScreen()

vUpdateNodeScreen()

vUpdateSensor()

vDrawGraph()

This group of functions build the display for the various display modes. They use sensor log data from the LogData component, outputting text and graphics to the LCD screen accordingly.

2.2.5 Log [app_log.c]

This component maintains data that stores the number of Sensor nodes and their addresses, and the sensor data itself. The number of Sensor nodes and their addresses are also persisted in Flash memory via the PDM.

APP_vLogInitialise()

This function sets the number of Sensor nodes to zero before attempting to load any existing record from Flash memory via the PDM.

APP_vLogStart()

This function starts a software timer that periodically calls the main logging task.

APP_psLogGetSensorNodeHistory()

APP_u8LogGetDataStartPos()

APP_u8GetSensorNodeId()

APP_u16GetSensorNodeAddr()

APP_u8ControllerNodeNumDataSensors()

This group of functions provides access to data structures held by the Log component.

OS_TASK(APP_taskLogData)

This task is responsible for registering new Sensor nodes when they first start transmitting data and for storing the sensor data that they send. It implements the SENSOR_DATA endpoint and receives the corresponding messages from the ZPS. The task is also periodically activated by a software timer that causes the task to advance the rolling data set (the last 16 sets of sensor data are stored for each Sensor node). In this case, the task updates the display after modifying the data set.

2.2.6 Start [app_start.c]

vAppMain()

This is the main code entry point after a reset or power-up. This function is responsible for initialising certain low-level hardware, such as the UART0 for debug and the CPU stack overflow monitor. It resets the IEEE 802.15.4 MAC layer and provides a trap for the watchdog reset event. It then starts the JenOS RTOS.

vAppRegisterPWRMCallbacks()

This function is used to register pre- and post-sleep callback functions with the Power Manager (PWRM) on Sleeping End Devices. This function must be present even if it is empty.

vInitialiseApp()

This function initialises several of the JenOS modules before calling the application's main initialisation function.

OS_ISR(APP_isrUnimplementedModuleException) **OS_ISR(APP_isrStackOverflowException)**

This is a pair of ISRs (Interrupt Service Routines) for handling CPU exceptions.

2.2.7 Sensor Node Components

A Sensor node consists of a number of components that correspond to individual source files. These implement the tasks and ISRs mentioned above in Sensor Node Architecture, and contain the various data structures along with access functions where appropriate, and additional low-level functions such as start-up routines and power management callbacks.

Table 6 below lists the components and corresponding filenames, and briefly describes its purpose. Note that the Router and SED types of Sensor node are largely the same. However, where different, the filenames for the SED-Sensor node are shown in brackets.

Component	File (.c/.h)	Purpose
Sensor Node	app_sensor_node app_sensor_node_SED	Contains the main SensorNode task, along with functions necessary to manage network discovery and joining. Responsible for initialisation of the ZigBee PRO stack and the application.
Sample	app_sample app_sample_SED	Contains the SampleSensors task. This component is responsible for initialising and controlling the light, temperature and humidity sensors in order to take sensor readings. It also generates SENSOR_DATA frames.
Buttons	app_buttons	Contains the ScanButtons task, which debounces button presses and posts appropriate messages to the ControllerNode task. Generates LED_CONTROL frames.
LED	app_led	Contains the LedControl task, which receives LED_CONTROL frames and switches LEDs on or off.
Start	app_start app_start_SED	Contains the entry point through which the device starts running after a reset or power-up and is responsible for starting the JenOS modules including the RTOS. Also contains some exception handlers and callbacks for the Power Manager.
System Controller	app_syscon	Contains the System Controller ISR. Identifies the interrupt source within the System Controller (DIO or Wake Timers) and activates the appropriate task.

Table 6: Sensor Node Components

The Sensor node components are described in detail below.

2.2.8 Sensor Node [app_sensor_node.c / _SED.c]

This component maintains a data structure that is responsible for the node's operational state. This data is registered with the JenOS Persistent Data Manager (PDM), and therefore persists through resets and power-cycles. The component contains the following tasks and functions:

APP_vInitialiseSensorNode()

This function is called during start-up. It initialises the hardware, the ZigBee PRO stack and the application. As part of this initialisation, the initial states of the buttons are checked. If any buttons are pressed at start-up, a call is made to the PDM requesting that the entire ZigBee PRO stack and application context is erased from Flash memory, thus causing a clean start. Otherwise, the function checks if a valid record has been retrieved from Flash memory by the PDM. If this is the case, the function re-starts the ZigBee PRO stack and application - the node is already joined to the network and does not need to perform the joining procedure. If no record is retrieved, the function starts the ZigBee PRO stack and application from fresh, causing the node to discover and join a network.

OS_TASK(APP_taskSensorNode)

This is the application's main task. It collects stack event messages and application messages, and then, depending on the application's state, calls one of a number of lower-level functions to handle the event.

vHandleStartupEvent()

This is the initial state. This function attempts to start the ZigBee PRO stack and then, if successful, advances the state machine to the next state. If unsuccessful, it activates the SensorNode task again which results in another attempt.

vHandleNetworkDiscoveryEvent()

This function waits for network discovery to complete. If successful and there is a suitable ZigBee PRO network for the node to join, the function initiates joining. If no suitable network is found, network discovery is initiated over the remaining unscanned channels. If network discovery fails, or if there are no unscanned channels remaining, then a software timer is started which re-activates the SensorNode task upon expiry. This causes the discovery and joining process to re-start from the beginning. If the parameter *apsUseExtendedPanID* is set to a non-zero value in the ZPS configuration file then the node will try to join a specific network using the Network Rejoin command. In this case, the ZigBee PRO stack performs discovery and rejoining as a single action, and then if successful, generates a stack event ZPS_EVENT_NWK_JOINED_AS_ROUTER. This causes the function to save the Sensor node's persistent data to Flash memory via the PDM and then advance the state machine to the E_MONITOR_SENSORS state without explicitly requesting joining.

vHandleNetworkJoinEvent()

This function waits for joining to complete. Then, if the attempt to join the network was successful, it switches the LEDs off to indicate that joining has finished, advances the state machine and then requests the PDM to save its persistent data record to Flash memory. On a Router, it starts a software timer in order to begin periodically sampling the sensors. On a SED, it activates the SampleSensors task directly.

vHandleMonitorSensorsEvent()

This function handles stack events when the network has formed. For most events, there is no action required. However, on a SED, when an APS Data Confirm is received, the application requests the JenOS Power Manager (PWRM) to schedule a sleep.

vHandleNetworkRejoinEvent()

This function starts a network discovery again and sets the state accordingly.

APP_u8GetSequenceNumber()

This function maintains the global APS data frame sequence number. It allows other components to access the sequence number when they require it, in order to send a data frame. The function optionally increments the sequence number.

vWakeCallBack()

This function is only present on a SED. It is called when the wake timer expires (regardless of whether the Power Manager had sent the unit to sleep). It restarts the SampleSensors task and initiates a poll request.

2.2.9 Sample [app_sample.c / _SED.c]

APP_vInitialiseSample()

This function initialises the Sensor node hardware.

OS_TASK(APP_taskSampleSensors)

This function contains a state machine which controls the sequence of events necessary to take light, temperature and humidity measurements from the sensor devices. On a Router, this task is activated periodically by a software timer. On a SED, it is activated each time the node wakes from sleep.

VSendSensorData()

This function builds a SENSOR_DATA frame and then passes it to the ZigBee PRO stack for transmission to the Controller.

OS_TASK(APP_taskSensorEvent)

This task is present only on Router-Sensor nodes. It receives messages associated with the SENSOR_DATA endpoint – specifically, it looks for APS Data Confirms that are generated when a SENSOR_DATA frame has been sent. When one is received, LED D2 is extinguished to indicate that the data transmission has completed.

2.2.10 Buttons [app_buttons.c]

APP_bButtonInitialise()

This function initialises the DIO lines that the buttons use, setting up levels and enabling interrupts appropriately.

OS_TASK(APP_taskScanButtons)

This task de-bounces button presses. It is activated 8 times by a software timer, once each time a button is pressed. After the final call, if the state of the button was constant each time the task was activated then a button event is generated. This is passed to **vHandleButtonEvent()**.

vHandleButtonEvent()

vSendLedData()

Between them, these two functions form a data frame containing LED_CONTROL data and then pass the frame to the ZigBee PRO stack for transmission to the Controller node.

2.2.11 LED [app_led.c]

APP_vLedsInitialise()

This function initialises the LEDs, ensures they are off, and disables the UART1's lines, which share DIO lines with the LEDs.

APP_vLedSet()

This function allows other components to switch LEDs on and off. This is used, for example, by the Sensor node task to switch all the LEDs off when the network has been joined.

OS_TASK(APP_taskLedControl)

This task receives messages associated with the LED_CONTROL endpoint. These contain frames from the Controller node and, according to their content, switch LED D1 on or off.

2.2.12 Start [app_start.c (app_start_SED.c)]

vAppMain()

This is the main code entry point after a reset or power-up. This function is responsible for initialising certain low-level hardware, such as the UART for debug and the CPU stack overflow monitor. It resets the IEEE 802.15.4 MAC layer and provides a trap for the watchdog reset event. It then starts the JenOS RTOS.

vAppRegisterPWRMCallbacks()

This function is used to register pre- and post-sleep callbacks with the Power Manager (PWRM) on Sleeping End Devices. The function must be present, even if empty.

vInitialiseApp()

This function initialises several of the JenOS modules before calling the Application's main initialisation function.

OS_ISR(APP_isrUnimplementedModuleException)

OS_ISR(APP_isrStackOverflowException)

This is a pair of ISRs (Interrupt Service Routines) for handling CPU exceptions.

PWRM_CALLBACK(PreSleep)

PWRM_CALLBACK(Wakeup)

This is a pair of callback functions which are present only on the Sleeping End Device.

- The first callback function is called before going to sleep and is responsible for saving the MAC settings to a RAM buffer. LED D2 is switched off to indicate that the device is sleeping.
- The second callback function is called upon waking, and is used to restore the IEEE 802.15.4 MAC settings, re-initialise the sensor hardware, and re-start the OS and application. LED D2 is switched on to indicate that the device is awake.

2.2.13 System Controller [app_syscon.c]

OS_ISR(APP_isrSysCon)

This ISR identifies the interrupt source within the System Controller (DIO or Wake Timers) and activates the appropriate task.

3 Re-building the Application

The ZigBee PRO Home Sensor Demonstration is supplied pre-built as binary files in the ZIP package of the Application Note. This chapter describes how to re-build the binaries, which you will need to do in either of the following cases:

- You wish to run an application binary on a JN5148 high-power module
- You have made alterations to the source code of the demonstration

If you are re-building to use the demonstration with high-power modules, first refer to Section 3.1. In either case, re-build as described in Section 3.2.

The instructions in this chapter assume that the Application Note's directory has been installed directly under **<JN5148_SDK_ROOT>\Application**, where **<JN5148_SDK_ROOT>** is the path into which the JN5148 SDK was installed (by default, this is **C:\Jennic**). The **Application** directory is automatically created when the SDK is installed.



Note: This application uses the ZigBee PRO wireless network protocol, which is only supported on the JN5148 device. Eclipse project files for the JN5148 device are provided.

3.1 Preparing for Use on High-Power Modules

To use an application with a JN5148 high-power module, a small modification must be made to the application source code and the application must be re-built.

The modification must be made in the source file **app_start.c**, which can be found in the **Source** sub-directory for the relevant application. Open this file and uncomment the following line in the **vAppMain()** function:


```
//vAHI_HighPowerModuleEnable(TRUE, TRUE);
```



Caution: This change must not be made when re-building an application for a standard-power module.

3.2 Building and Downloading the Application

To build the application in the Eclipse IDE and load the resulting binaries into the JN5148 boards, follow the instructions below:

1. Ensure that the project directory is located in
<JN5148_SDK_ROOT>\Application
where **<JN5148_SDK_ROOT>** is the path into which the SDK was installed.
2. Start the Eclipse platform and import the relevant project files (**.project** and **.cproject**) as follows:
 - a) In Eclipse, follow the menu path **File>Import** to display the Import dialogue box.
 - b) In the dialogue box, expand **General** and select **Existing Projects into Workspace**.
 - c) Enable **Select root directory** and browse to the **Application** directory.
 - d) In the **Projects** box, select the project to be imported.
3. Build the project. To do this, use the drop-down list associated with the hammer icon  in the Eclipse toolbar to select the relevant build configuration – once selected, the project will automatically build.
The binary files will be created in the relevant build configuration directory.
4. Load the resulting binary files into the boards from the appropriate build configuration directory. You can do this using the JN51xx Flash Programmer (described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*), which can be launched either directly or from within Eclipse.

Revision History

Version	Description
1.0	First release
1.1	Added note about handling network joins in application – application code also modified accordingly
2.0	Demo operational information from JN5148-EK010 Evaluation Kit User Guide (JN-UG-3062) migrated into Application Note

Important Notice

Jennic reserves the right to make corrections, modifications, enhancements, improvements and other changes to its products and services at any time, and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders, and should verify that such information is current and complete. All products are sold subject to Jennic's terms and conditions of sale, supplied at the time of order acknowledgment. Information relating to device applications, and the like, is intended as suggestion only and may be superseded by updates. It is the customer's responsibility to ensure that their application meets their own specifications. Jennic makes no representation and gives no warranty relating to advice, support or customer product design.

Jennic assumes no responsibility or liability for the use of any of its products, conveys no license or title under any patent, copyright or mask work rights to these products, and makes no representations or warranties that these products are free from patent, copyright or mask work infringement, unless otherwise specified.

Jennic products are not intended for use in life support systems/appliances or any systems where product malfunction can reasonably be expected to result in personal injury, death, severe property damage or environmental damage. Jennic customers using or selling Jennic products for use in such applications do so at their own risk and agree to fully indemnify Jennic for any damages resulting from such use.

All trademarks are the property of their respective owners.

NXP Laboratories UK Ltd

(Formerly Jennic Ltd)

Furnival Street

Sheffield

S1 4QT

United Kingdom

Tel: +44 (0)114 281 2655

Fax: +44 (0)114 281 2951

E-mail: info@jennic.com

For the contact details of your local Jennic office or distributor, refer to the Jennic web site:

www.nxp.com/jennic