



# Application Note: JN-AN-1184

## ZigBee PRO Application Template for JN516x

---

This Application Note provides a set of ZigBee PRO application code templates based on the NXP ZigBee PRO and JenOS APIs. These templates are sufficient to produce a set of null network nodes - that is, nodes which are able to start or join a ZigBee PRO network but have no other application running on them. The templates therefore provide generic code which can be used as the basis for application development.

Three node types are provided:

- Co-ordinator
- Router
- Sleeping End Device

To use the application template, you must have the 'BeyondStudio for NXP' toolchain (JN-SW-4141) and JN516x ZigBee Light Link/Home Automation SDK (JN-SW-4168), which can be obtained via the [NXP Wireless Connectivity TechZone](#).

---

## Application Overview

The ZigBee PRO application templates contain code which builds into a set of null nodes that provide the minimum functionality for the three node types (Co-ordinator, Router, Sleeping End Device) to initialise themselves and then start or join a network

The templates show how to:

- initialise the IEEE 802.15.4 MAC layer, the ZigBee PRO stack and the JenOS modules
- deal with stack and application persistence using the Persistent Data Manager (so that the node state is maintained through resets/power-cycles)
- configure a Sleeping End Device to sleep and poll its parent (for data) on waking
- implement an endpoint as a task for receiving frames and APS confirmations associated with that endpoint



**Note 1:** 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 included in the code for this Application Note.



**Note 2:** For details of the ZigBee PRO and JenOS APIs, you should refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)* and *JenOS User Guide (JN-UG-3075)*, respectively. Both manuals are available from the [NXP Wireless Connectivity TechZone](#).

## Co-ordinator

The Co-ordinator application contains the following components (each corresponding to a source file):

### **app\_coordinator.c/.h**

This code implements an initialisation function along with a single task, **APP\_taskCoordinator**. The initialisation function handles registering and loading any persistent application data using the JenOS PDM, and initialisation of the ZigBee PRO stack. The task provides a state machine which starts the node as a ZigBee Co-ordinator, calling a number of lower level functions to enable this. If the node has been configured with the network parameter *apsUseExtendedPanID* set to zero (using the ZPS Configuration Editor), the device uses its MAC address as the Extended PAN ID (EPID) - otherwise, it uses the EPID specified by *apsUseExtendedPanID*. Once running, the node becomes idle, with the ZigBee PRO stack automatically responding to any network events.

### **app\_endpoint.c**

This code implements an endpoint as a task. This receives messages posted from the APS when a frame is received on this endpoint, and also receives any confirmations generated as a result of frames being sent from this endpoint.

### **app\_start.c**

This code provides the initial entry point when the device powers up or comes out of reset. It initialises certain low-level hardware, such as the UART for debug and the CPU stack overflow monitor. It also resets the IEEE 802.15.4 MAC layer and provides a trap for the watchdog reset event. It then starts the RTOS and other JenOS modules before calling the application initialisation function.

## Router

The Router application contains the following components (each corresponding to a source file):

### **app\_router.c/.h**

This code implements an initialisation function along with a single task, **APP\_taskRouter**. The initialisation function handles registering and loading any persistent application data using the JenOS PDM, and initialisation of the ZigBee PRO stack. The task provides a state machine which starts the node as a ZigBee Router, calling a number of lower level functions to enable this. If the node has been configured with the network parameter *apsUseExtendedPanID* set to zero (using the ZPS Configuration Editor) then the node will search for suitable ZigBee PRO networks that it may join and then attempt to join one of them - otherwise, it will attempt to rejoin the network with EPID specified by *apsUseExtendedPanID*. Once running, the node becomes idle, with the ZigBee PRO stack automatically responding to any network events.

### **app\_endpoint.c**

This code implements an endpoint as a task. This receives messages posted from the APS when a frame is received on this endpoint, and also receives any confirmations generated as a result of frames being sent from this endpoint.

### **app\_start.c**

This code provides the initial entry point when the device powers up or comes out of reset. It initialises certain low-level hardware, such as the UART for debug and the CPU stack overflow monitor. It also resets the IEEE 802.15.4 MAC layer and provides a trap for the

watchdog reset event. It then starts the RTOS and other JenOS modules before calling the application initialisation function.

## Sleeping End Device

The Sleeping End Device application contains the following components (each corresponding to a source file):

### **app\_sleeping\_enddevice.c/h**

This code implements an initialisation function along with a single task, **APP\_taskSleepingEndDevice**. The initialisation function handles registering and loading any persistent application data using the JenOS PDM, and initialisation of the ZigBee PRO stack. The task provides a state machine which starts the node as a ZigBee Sleeping End Device, calling a number of lower level functions to enable this. If the node has been configured with the network parameter *apsUseExtendedPanID* set to zero (using the ZPS Configuration Editor) then the node will search for suitable ZigBee PRO networks that it may join and then attempt to join one of them - otherwise, it will attempt to rejoin the network with EPID specified by *apsUseExtendedPanID*. Once joined, the node will schedule sleep using the Power Manager (PWRM) module - by default, the sleep duration is set to 5 seconds. On wake-up, the **vWakeCallback()** callback function is called, which requests the device to poll its parent for data. The device resumes running. On receipt of a Poll Confirm stack event, the device schedules sleep again. This process then repeats.

### **app\_endpoint.c**

This code implements an endpoint as a task. This receives messages posted from the APS when a frame is received on this endpoint, and also receives any confirmations generated as a result of frames being sent from this endpoint.

### **app\_start\_SED.c**

This code provides the initial entry point when the device powers up or comes out of reset. It initialises certain low-level hardware, such as the UART for debug and the CPU stack overflow monitor. It also resets the IEEE 802.15.4 MAC layer and provides a trap for the watchdog reset event. It then starts the RTOS and other JenOS modules before calling the application initialisation function. It also provides pre- and post-sleep callback functions for the Power Manager, which saves the MAC radio settings before sleeping, and restores and restarts the RTOS on waking. These functions may be used to switch off any hardware before sleeping (in order to minimise power consumption during sleep) and to re-initialise any hardware on waking.

### **app\_syscon.c**

This code provides the System Controller ISR (Interrupt Service Routine) which handles the interrupts generated by the Wake Timer used for sleep.

## Customising the Template

The Project can be renamed by right-clicking on it in the **Projects** view in 'BeyondStudio for NXP' and selecting **Rename**.

Source files can be renamed by right-clicking the file in the **Projects** view and selecting **Rename**. The Application Source section of the associated makefile must then be edited to reflect the new name.

To change the name of the application binary file which will result from a build, edit the makefile and change the Target definition:

```
TARGET = myTargetName
```

To add a new source file, follow the procedure below:

1. Right-click on the **<Device\_type>\Source** folder in the **Projects** view, e.g. to add a new Co-ordinator source file, right-click on **Coordinator\Source**.
2. Select **New > Source File**. This causes a **New Source File** window to appear.
3. Enter the source file name in the **New Source File** window.
4. Click **Finish**. The new source file appears in the **Project Explorer** panel.
5. Edit the associated makefile and add the new source file to the Application Source section:


```
APPSRC += myNewSource.c
```

## Building and Downloading the Application

It is assumed that you have installed the 'BeyondStudio for NXP' toolchain (JN-SW-4141) and JN516x ZigBee Light Link/Home Automation SDK (JN-SW-4168) on your PC. You can obtain the latest versions via the [NXP Wireless Connectivity TechZone](#).

In order to build the software provided with this Application Note, the application's folder must be placed directly under **<BeyondStudio for NXP installation root>\workspace**, where **<BeyondStudio for NXP installation root>** is the path into which BeyondStudio for NXP was installed (by default, this is **C:\NXP\bstudio\_nxp**). The **workspace** directory is automatically created when you start BeyondStudio for NXP.

To build the application and load it into the JN516x boards, follow the instructions below:

1. Ensure that the project directory is located in  
**<BeyondStudio for NXP installation root>\workspace**
2. Start the BeyondStudio for NXP and import the relevant project as follows:
  - a) In BeyondStudio, follow the menu path **File>Import** to display the **Import** dialogue box.
  - b) In the dialogue box, expand **General**, select **Existing Projects into Workspace** and click **Next**.
  - c) Enable **Select root directory** and browse to the **workspace** directory.
  - d) In the **Projects** box, select the project to be imported and click **Finish**.
3. Build an application. To do this, ensure that the project is highlighted in the left panel of BeyondStudio and use the drop-down list associated with the hammer icon  in the toolbar to select the relevant build configuration – once selected, the application will automatically build. Repeat this to build the other applications.  
 The binary files will be created in the relevant **Build** directories for the applications.
4. Load the resulting binary files into the board. You can do this using the integrated Flash programmer, as described in the *BeyondStudio for NXP Installation and User Guide* (JN-UG-3098).

## Revision History

Version	Notes
1.0	First release
1.1	Software updated
1.2	Updated to be compatible with the 'BeyondStudio for NXP' toolchain (JN-SW-4141)

## Important Notice

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

All trademarks are the property of their respective owners.

## NXP Semiconductors

For the contact details of your local NXP office or distributor, refer to:

[www.nxp.com](http://www.nxp.com)