Application Note: JN-AN-1229

ZigBee PRO Application Template for ZigBee 3.0

This Application Note provides a set of ZigBee PRO application code templates based on the NXP ZigBee PRO stack for ZigBee 3.0 on the JN516x and JN517x wireless microcontrollers. 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 JN516x ZigBee 3.0 SDK (JN-SW-4170) or JN517x ZigBee 3.0 SDK (JN-SW-4270) and relevant toolchain, which can be obtained via the ZigBee 3.0 page of the NXP web site.

1 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
- 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 stack, you should refer to the *ZigBee 3.0 Stack User Guide (JN-UG-3113)*.

2 Development Environment

2.1 Software

In order to use this Application Note, you need to install the Eclipse-based Integrated Development Environment (IDE) and Software Developer's Kit (SDK) that are appropriate for the chip family which you are using - either JN516x or JN517x:

- **JN516x:** If developing for the JN516x microprocessors, you will need:
  - 'BeyondStudio for NXP' IDE [JN-SW-4141]
  - JN516x ZigBee 3.0 SDK [JN-SW-4170]

  For installation instructions, refer to the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.
- **JN517x:** If developing for the JN517x microprocessors, you will need:
  - LPCXpresso IDE
  - JN517x ZigBee 3.0 SDK [JN-SW-4270]

  For installation instructions, refer to the *JN517x LPCXpresso Installation and User Guide (JN-UG-3109)*.

The LPCXpresso software can be obtained as described in the *JN517x ZigBee 3.0 SDK Release Notes*, which indicate the version that you will need.

All other resources are available via the **ZigBee 3.0** page of the NXP web site.

Note: The code in this Application Note can be used in either BeyondStudio or LPCXpresso and the process for importing the application into the development workspace is the same for both.

Note: Prebuilt JN5169 and JN5179 application binaries are supplied in this Application Note package, but the applications can be rebuilt for other devices in the JN516x and JN517x families (see Section 6).

2.2 Hardware

Hardware kits are available from NXP to support the development of ZigBee 3.0 applications. The following kits respectively provide JN516x-based and JN517x-based platforms for running these applications:

- JN516x-EK004 Evaluation Kit, which features JN5169 devices
- JN517x-DK005 Development Kit, which features JN5179 devices

It is also possible to develop ZigBee 3.0 applications to run on the components of the earlier JN516x-EK001 Evaluation Kit, which features JN5168 devices.

3 Capabilities

This Application Note is designed for use with the following NXP hardware and software:

Product Type	Part Number
Evaluation/Development Kit	JN516x-EK004 JN517x-DK005
JN516x ZigBee 3.0 SDK (BeyondStudio only)	JN-SW-4170
JN517x ZigBee 3.0 SDK (LPCXpresso only)	JN-SW-4270
'BeyondStudio for NXP' Toolchain	JN-SW-4141
LCPXpresso Toolchain	7.9.2 (Build 493)

4 Application Code

This section lists and describes the application files for the ZigBee devices (Co-ordinator, Router and Sleeping End Device).

4.1 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 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.

4.2 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 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.

4.3 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 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 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.

5 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
```

6 Building and Downloading the Application

This section describes how to build the supplied applications for JN516x and JN517x devices.

6.1 Pre-requisites

It is assumed that you have installed the relevant NXP development software on your PC, as detailed in Section 2.

In order to build the application, this Application Note [JN-AN-1229] must be unzipped into the directory:

<IDE installation root>\workspace

where **<IDE Installation root>** is the path in which the IDE was installed. By default, this is:

- **C:\NXP\bstudio_nxp** for BeyondStudio
- **C:\NXP\LPCXpresso_<version>_<build>\lpcxpresso** for LPCXpresso

The **workspace** directory is automatically created when you start the IDE.

All files should then be located in the directory:

…\workspace\JN-AN-1229-ZBPro-Application-Template

There is a sub-directory for each application, each having **Source** and **Build** sub-directories. There will also be sub-directories **JN516x** and **JN517x** containing the project definition files.

6.2 Build Instructions

The software provided with this Application Note can be built for the JN516x and JN517x devices.

The applications can be built from the command line using makefiles or from the IDE (BeyondStudio or LPCXpresso) – makefiles and Eclipse-based project files are supplied.

- To build using makefiles, refer to Section 6.2.1.
- To build using the IDE, refer to Section 6.2.2.

6.2.1 Using Makefiles

This section describes how to use the supplied makefiles to build the applications. Each application has its own **Build** directory, which contains the makefiles for the application.

To build an application and load it into a JN516x/7x board, follow the instructions below:

1. Ensure that the project directory is located in

<IDE installation root>\workspace

2. Start an MSYS shell by following the Windows Start menu path:
All Programs > NXP > MSYS Shell

3. Navigate to the **Build** directory for the application to be built and at the command prompt enter an appropriate `make` command for your chip type, as illustrated below.

For example, for JN5169:

```
make JENNIC_CHIP_FAMILY=JN516x JENNIC_CHIP=JN5169 clean all
```

For example, for JN5179:

```
make JENNIC_CHIP_FAMILY=JN517x JENNIC_CHIP=JN5179 clean all
```

The binary file will be created in the **Build** directory, the resulting filename indicating the chip type (e.g. **5169**) for which the application was built.

4. Load the resulting binary file into the board. You can do this from the command line using the JN51xx Production Flash Programmer (JN-SW-4107), described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099)*.

6.2.2 Using the IDE (BeyondStudio for NXP or LPCXpresso)

This section describes how to use the IDE to build the demonstration application.

To build the application and load it into JN516x/7x boards, follow the instructions below:

1. Ensure that the project directory is located in

<IDE installation root>\workspace

2. Start the IDE and import the relevant project as follows:

a) In the IDE, 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, only select the project file appropriate for the chip family and IDE that you are using, and click **Finish**.

3. Build an application. To do this, ensure that the project is highlighted in the left panel of the IDE 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 User Guide for the IDE that you are using.

7 Release Details

7.1 Tested With

Product Type	Part Number	Build
Version 1001		
JN516x ZigBee 3.0 SDK	JN-SW-4170	1518
JN517x ZigBee 3.0 SDK	JN-SW-4270	1520
‘BeyondStudio for NXP’ Toolchain	JN-SW-4141	1308
LPCXpresso Toolchain	7.9.2	493

7.2 New Features

ID	Feature	Description
Version 1002		
None		
Versions 1001 and 1000		
None		

7.3 Known Issues

ID	Severity	Description
Version 1002		
None		
Versions 1001 and 1000		
None		

7.4 Bug Fixes

ID	Description
Version 1002	
None	
Versions 1001 and 1000	
None	

Revision History

Version	Notes
1000	First release
1001	Updated for the JN517x devices
1002	Software updated in line with latest builds of the ZigBee 3.0 SDKs (see Section 7)

Important Notice

NXP Semiconductors

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

www.nxp.com