



# **IEEE 802.15.4 Application Development Reference Manual**

JN-RM-2024  
Revision 2.0  
18-Jun-2010



---

## Contents

<b>About this Manual</b>	<b>4</b>
Organisation	4
Conventions	4
Acronyms and Abbreviations	5
Related Documents	5
Feedback Address	5
<b>1 Software Overview</b>	<b>7</b>
1.1 Software Architecture	7
1.2 Application Programming Interfaces (APIs)	8
1.2.1 802.15.4 Stack API	8
1.2.2 Integrated Peripherals API	8
1.2.3 Board API	8
1.2.4 Application Queue API (Optional)	8
1.3 Interrupts and Callbacks	9
<b>2 Network Set-up Process</b>	<b>11</b>
2.1 Overview	11
2.2 Stages of Set-up Process	12
2.2.1 Initialising the Stack	12
2.2.2 Creating a PAN Co-ordinator	12
2.2.3 Selecting the PAN ID and Co-ordinator Short Address	12
2.2.4 Selecting a Radio Frequency	13
2.2.5 Starting the Network	13
2.2.6 Joining Devices to the Network	13
2.2.7 Transferring Data between Devices	14
<b>3 Easy Application Development</b>	<b>17</b>
3.1 Application Template	17
3.1.1 Pre-requisites	17
3.1.2 Unpacking the Application Note	18
3.1.3 Supplied Files	18
3.2 Code Descriptions	19
3.2.1 Contents of AN1046_154_Coord.c	20
3.2.2 Contents of AN1046_154_EndD.c	22
3.3 Adapting the Skeleton Code	25
3.3.1 How Do I Program a Pre-defined PAN ID?	25
3.3.2 How Do I Program Pre-defined Short Addresses?	25
3.3.3 How Do I Add End Devices to the Network?	26
3.3.4 How Do I Program the Channel Scans?	26
3.3.5 How Do I Define the Processing of Received Data Packets?	28
3.3.6 How Do I Program Data Transmission?	29
<b>4 Compiling Your Application Code</b>	<b>31</b>
4.1 Building Your Code	31
4.1.1 Building Code Using Makefiles	31
4.1.2 Building Code Using Eclipse (JN5148 Only)	32
4.1.3 Building Code Using Code::Blocks (JN5139/JN5121 Only)	33

---

## About this Manual

This manual supports the software template provided by Jennic for the development of applications to be deployed in an IEEE 802.15.4-based wireless network. The template is relevant to non-beacon enabled networks only. This manual describes the tasks undertaken in the supplied skeleton code and the function calls employed. The template along with this manual should enable you to streamline your application development and rapidly achieve effective IEEE 802.15.4 applications.

The template is available for download from the Application Notes section of the Support area of the Jennic web site ([www.jennic.com/support](http://www.jennic.com/support)). The part number is JN-AN-1046.



**Note:** If you are not familiar with the basics of the IEEE 802.15.4 standard, you should first refer to the *Jennic IEEE 802.15.4 Wireless Networks User Guide (JN-UG-3024)* before attempting application development.

---

## Organisation

This document consists of four chapters, as follows:

- [Chapter 1](#) provides an overview of the IEEE 802.15.4 software in terms of architecture, interfaces and interrupts.
- [Chapter 2](#) describes the process of setting up a simple IEEE 802.15.4 based network.
- [Chapter 3](#) describes the skeleton application code supplied by Jennic and also provides guidelines on adapting this code.
- [Chapter 4](#) details how to compile your application code.

---

## 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

FFD	Full Function Device
IDE	Integrated Development Environment
MAC	Media Access Control (sub-layer)
PAN	Personal Area Network
PHY	Physical (layer)
RFD	Reduced Function Device

---

## Related Documents

- [1] IEEE 802.15.4 Standard – 2003 [SS95127] (URL: <http://www.ieee.com/>)
- [2] Jennic IEEE 802.15.4 Wireless Networks User Guide [JN-UG-3024]
- [3] Jennic Integrated Peripherals API Reference Manual [JN-RM-2001]
- [4] Jennic 802.15.4 Stack API Reference Manual [JN-RM-2002]
- [5] Jennic Board API Reference Manual [JN-RM-2003]
- [6] Jennic Application Queue API Reference Manual [JN-RM-2025]
- [7] Jennic Code::Blocks IDE User Guide [JN-UG-3028]
- [8] Jennic JN51xx Flash Programmer Application User Guide [JN-UG-3007]

---

## Feedback Address

If you wish to comment on this manual, or any other Jennic user documentation, 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  
Jennic Ltd  
Furnival Street  
Sheffield S1 4QT  
United Kingdom  
[doc@jennic.com](mailto:doc@jennic.com)

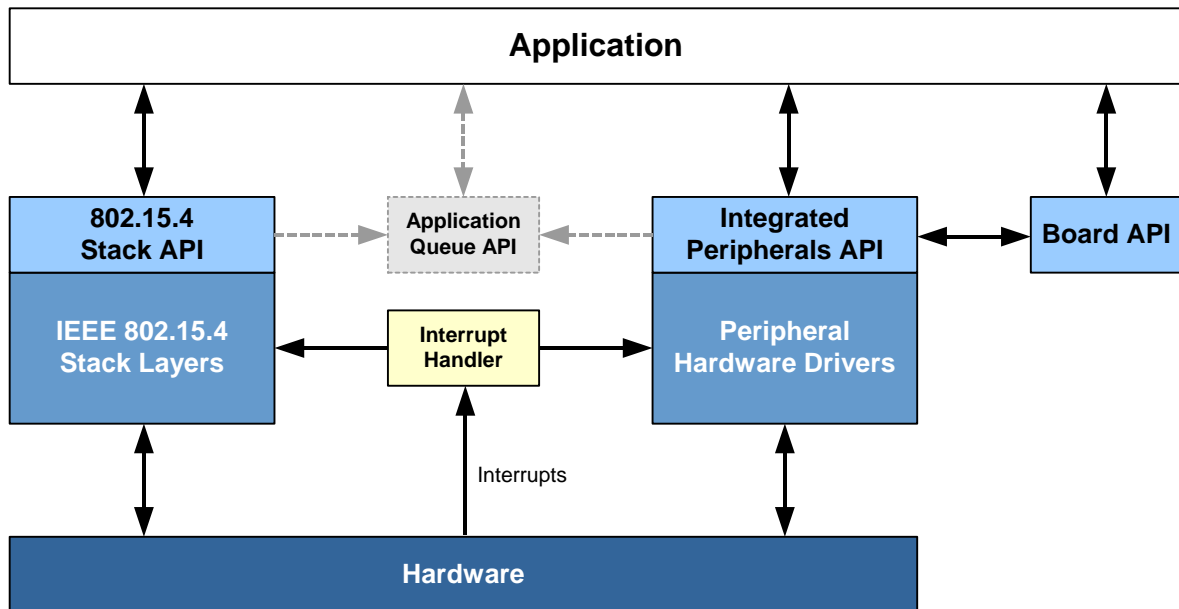


# 1 Software Overview

This chapter provides the essential software concepts in an IEEE 802.15.4 based wireless network that uses the Jennic JN51xx wireless microcontroller.

## 1.1 Software Architecture

The basic software architecture in which the application sits is illustrated and described below.



**Figure 1: Software Architecture**

The main features of this architecture are as follows:

- The application uses functions of the 802.15.4 Stack API to interact with the IEEE 802.15.4 stack layers. This interaction is implemented in terms of MCPS/MLME requests and confirmations, indications and responses. The IEEE 802.15.4 stack interacts with the underlying hardware to access hardware registers.
- The application interacts with the on-chip hardware peripherals using functions of the Integrated Peripherals API. This API uses the peripheral hardware drivers to access hardware registers.
- The application interacts with the (evaluation kit) board hardware peripherals using functions of the Board API. The Board API uses the Integrated Peripherals API to achieve the interaction with the board hardware.
- The hardware generates interrupts which are routed to the appropriate software block (IEEE 802.15.4 stack or peripheral hardware drivers) by an interrupt handler.
- Optionally, the Application Queue API can be used to lighten the application's involvement in dealing with interrupts.

---

## 1.2 Application Programming Interfaces (APIs)

This section outlines the Application Programming Interfaces (APIs) used by an IEEE 802.15.4 application.

---

### 1.2.1 802.15.4 Stack API

The Jennic 802.15.4 Stack API allows the application to interact with the IEEE 802.15.4 stack by facilitating control of the IEEE 802.15.4 MAC hardware on the Jennic JN51xx wireless microcontroller.

This API is described in the *802.15.4 Stack API Reference Manual (JN-RM-2002)*.

---

### 1.2.2 Integrated Peripherals API

The Jennic Integrated Peripherals API allows the application to create, control and respond to events in the peripheral blocks of the Jennic JN51xx wireless microcontroller (e.g. UARTs, timers and GPIOs).

This API is described in the *Integrated Peripherals API Reference Manual (JN-RM-2001)*.

---

### 1.2.3 Board API

The Jennic Board API allows the application to control the peripherals on boards carrying the Jennic JN51xx wireless microcontroller. These peripherals may include LCD panels, LEDs and buttons, as well as temperature, humidity and light sensors. The API allows the easy manipulation of hardware registers.

This API is described in the *Board API Reference Manual (JN-RM-2003)*.

---

### 1.2.4 Application Queue API (Optional)

Use of the Jennic Application Queue API is optional. This API handles all interrupts by providing a queue-based interface, saving the application from dealing with interrupts directly. When an interrupt is generated, an entry is placed in one of three queues (corresponding to MLME, MCPS and hardware events). The application can then poll the queues for events and deal with them when convenient.

The Application Queue API allows callbacks to be defined by the application, similar to the normal 802.15.4 Stack API, but an application can be designed such that they are not necessary.

This API is described in the *Application Queue API Reference Manual (JN-RM-2025)*.



**Note:** This API will be useful in some applications, but it is not essential for operation of the stack.



---

## 1.3 Interrupts and Callbacks



**Note:** This section is not applicable if you are using the Application Queue API to handle interrupts (see Section 1.2.4).

Any call into the stack through an API entry point is performed in the application task context.

Many of the possible 802.15.4 requests cause the stack to initiate activities that will continue after the call has returned, such as a request to transmit a frame. In such cases, the stack will acquire processor time by responding to interrupts from the hardware. To avoid the need for a multi-tasking operating system, the stack will then work for as long as necessary in the interrupt context.

When information has to be sent to the application, either because of a previous request or due to an indication from the stack or hardware, the appropriate callback function is used. It must be remembered that the callback is still in the interrupt context and that any activity performed by the application within the callback must be kept as short as possible.

All interrupts are generated by hardware. An interrupt handler in software decides whether to pass each interrupt to the 802.15.4 stack or to the peripheral hardware drivers. These either process the interrupt themselves or pass it up to the application via one of the registered callbacks.



## 2 Network Set-up Process

This chapter outlines the tasks that an application must go through in order to get an IEEE 802.15.4 based network up and running. The assumed topology is a Star network. Note that the application described in this document is for a **non-beacon enabled network only**.

### 2.1 Overview

The flowchart below provides an overview of the steps in setting up an IEEE 802.15.4 based network.

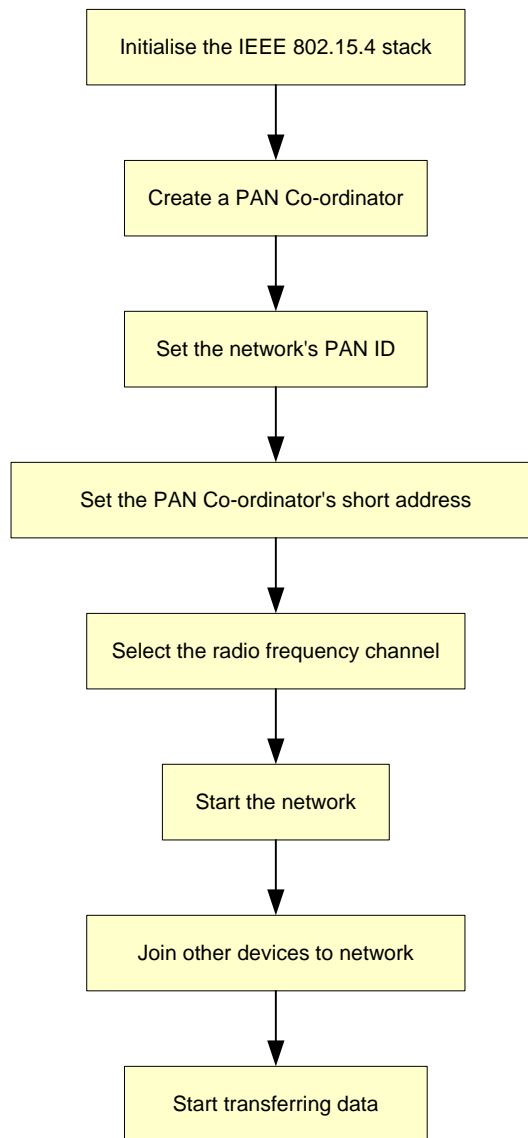


Figure 2: Network Set-up Process

---

## 2.2 Stages of Set-up Process

The steps indicated in the above flowchart are expanded on in the sections below.



**Note:** The process described here assumes that the device that is to become the PAN Co-ordinator has been pre-determined. The PAN Co-ordinator must be a Full Function Device (FFD). All Jennic devices are FFDs.

---

### 2.2.1 Initialising the Stack

First of all, the PHY and MAC layers of the IEEE 802.15.4 stack must be initialised on each device which will form part of the network.

---

### 2.2.2 Creating a PAN Co-ordinator

Every network must have one and only one PAN Co-ordinator, and one of the first tasks in setting up a network is to select and initialise this Co-ordinator. This involves activity only on the device nominated as the PAN Co-ordinator.

---

### 2.2.3 Selecting the PAN ID and Co-ordinator Short Address

The PAN Co-ordinator must assign a PAN ID to its network. The PAN ID may be pre-determined.



**Note:** The PAN Co-ordinator can choose a PAN ID automatically by 'listening' for other networks and selecting a PAN ID that does not conflict with the IDs of any existing networks that it detects. It can perform this scan for other PAN Co-ordinators over multiple radio frequency channels. Alternatively, a radio frequency channel for the network can be chosen first and the PAN ID then selected according to other PAN IDs detected in this channel - in this case, the step described in Section 2.2.4 must be performed first.

The PAN Co-ordinator device already has a fixed 64-bit IEEE (MAC) address, sometimes called the 'extended' address, but must also assign itself a local 16-bit network address, usually called the 'short' address. Use of the short address makes communications lighter and more efficient. This address is pre-determined; the PAN Co-ordinator is usually assigned the short address 0x0000.

---

## 2.2.4 Selecting a Radio Frequency

The PAN Co-ordinator must select the radio frequency channel in which the network will operate, within the chosen frequency band. The PAN Co-ordinator can select the channel by performing an Energy Detection Scan in which it scans the frequency channels to find a quiet channel. The Co-ordinator can be programmed to only scan specific channels. The Energy Detection Scan returns an energy level for each channel scanned, which indicates the amount of activity on the channel. The application running on the PAN Co-ordinator must then choose a channel using this information.

---

## 2.2.5 Starting the Network

The network is started by first completing the configuration of the device which will act as the PAN Co-ordinator and then starting the device in Co-ordinator mode. The PAN Co-ordinator is then open to requests from other devices to join the network.

---

## 2.2.6 Joining Devices to the Network

Other devices can now request to join the network. A device wishing to join the network must first be initialised and must then find the PAN Co-ordinator.

To find the PAN Co-ordinator, the device performs an Active Channel Scan in which it sends out beacon requests across the relevant frequency channels. When the PAN Co-ordinator detects the beacon request, it responds with a beacon to indicate its presence to the device.



**Note:** In the case of a beacon enabled network (in which the PAN Co-ordinator sends out periodic beacons), the device can perform a Passive Channel Scan in which the device 'listens' for beacons from the PAN Co-ordinator in the relevant frequency channels. However, this method is beyond the scope of this document.

Once the device has detected the PAN Co-ordinator, it sends an association request to the Co-ordinator, which acknowledges the request. The Co-ordinator then determines whether it has the resources to support the new device and either accepts or rejects the device.

If the PAN Co-ordinator accepts the device, it may assign a 16-bit short address to the device.

## 2.2.7 Transferring Data between Devices

Once equipped with a PAN Co-ordinator and at least one other device, the network is ready to exchange data. The scenarios for transferring data are outlined below.

### 'Co-ordinator to End Device' Transfers

Two methods of data transfer from Co-ordinator to End Device are available:

- **Direct Transmission:** The PAN Co-ordinator can send a data frame directly to an End Device in its network. Once it has received the data, the End Device can send an acknowledgement to the Co-ordinator. In this case, the End Device must always be capable of receiving data and must therefore be permanently active. This approach is employed in the skeleton code described in this document.
- **Indirect Transmission:** Alternatively, the PAN Co-ordinator can hold onto data until the data is requested by the relevant End Device. In this case, in order to obtain data from the PAN Co-ordinator, an End Device must first poll the Co-ordinator to determine whether any data is available. To do this, the device sends a data request, which the Co-ordinator acknowledges. The Co-ordinator then determines whether it has any data for the requesting device; if it does, it sends a data packet, which the receiving device may acknowledge. This method is useful when the End Device is a low-power device that must sleep for much of the time in order to conserve power.

The above two data transfer methods are illustrated in the figure below.

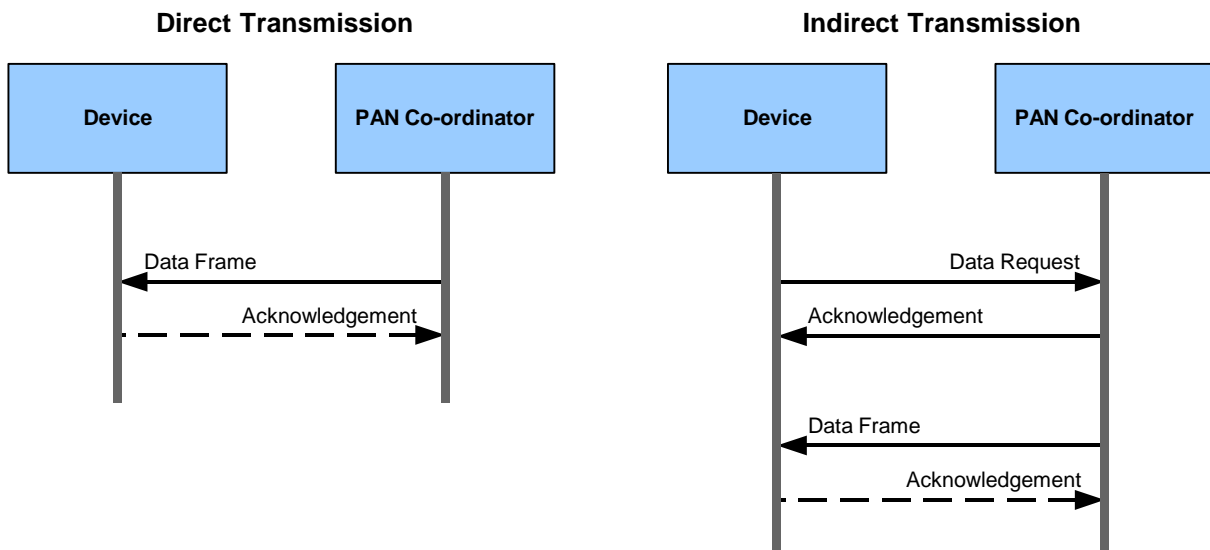


Figure 3: 'Co-ordinator to End Device' Data Transfers

### 'End Device to Co-ordinator' Transfers

An End Device always sends a data frame directly to the PAN Co-ordinator (without polling). Once it has received the data, the Co-ordinator may send an acknowledgement to the End Device.



**Note:** A data frame can be broadcast to all nodes within range and operating in the same network (i.e. using the same PAN ID) by setting the destination (short) address in the frame to 0xFFFF. Alternatively, a data frame can be broadcast to all nodes within range and operating in any network by setting the destination PAN ID in the frame to 0xFFFF and the destination (short) address to 0xFFFF.





---

## 3 Easy Application Development

This chapter describes the application template which this manual accompanies. The supplied files are described first and then the function calls within the code.

---

### 3.1 Application Template

The application template provides a basis for your own application development for an IEEE 802.15.4 based wireless network (non-beacon enabled). You can modify the supplied code to adapt it to your own application needs.

The template is available for download from the Application Notes section of the Support area of the Jennic web site. The part number is JN-AN-1046.

---

#### 3.1.1 Pre-requisites

It is assumed that you have installed the relevant Jennic SDK on your PC – the required SDK installers depend on your chip type, as follows:

- **JN5148:** JN-SW-4040 and JN-SW-4041
- **JN5139/JN5121:** JN-SW-4030 and JN-SW-4031

The above installers are available from the Support area of the Jennic web site ([www.jennic.com/support](http://www.jennic.com/support)). When installing the SDK, follow the instructions provided in the appropriate SDK Installation Guide: JN-UG-3064 for JN5148, JN-UG-3035 for JN5139/JN5121.

The skeleton application in the Application Note (JN-AN-1046) assumes the following:

- You have one device which will act as the PAN Co-ordinator.
- You have at least one other device which will act as an End Device.
- You will use pre-determined values for the PAN ID and the short addresses (for the PAN Co-ordinator and for the End Device(s)).
- The network topology will be a Star network.
- The network will be non-beacon enabled (meaning that the PAN Co-ordinator will not transmit regular beacons).
- Short addressing will be used.
- Data transfers will be direct transmissions with acknowledgements.
- There will be no security implemented.

---

### 3.1.2 Unpacking the Application Note

Unzip the JN-AN-1046 Application Note into the **Application** directory of the Jennic SDK installation. The path of this directory depends on the chip type, as follows:

- JN5148: <JENNIC\_SDK\_ROOT>\Application
- JN5139/JN5121: <JENNIC\_SDK\_ROOT>\cygwin\jennic\SDK\Application

where <JENNIC\_SDK\_ROOT> is the path into which the Jennic SDK was installed (by default, this is **C:\Jennic**). The **Application** directory is automatically created when you install the Jennic SDK.

Ensure that the created folder **JN-AN-1046-802-15-4-App-Template** is directly under **Application**. You should rename the Application Note folder with the name of your project.

---

### 3.1.3 Supplied Files

The application's file structure includes the following folders:

- **AN1046\_154\_Coord** – contains source files and makefiles for the PAN Co-ordinator
- **AN1046\_154\_EndD** – contains source files and makefiles for an End Device
- **Common** – contains the **config.h** header file used for both devices, which defines certain values used in the source code (e.g. PAN ID, short addresses, channels to scan)
- **CodeBlocksProject** – contains Code::Blocks project files for both devices (relevant only when developing for JN5139/JN5121 in the Code::Blocks IDE)
- **EclipseDebugConfig** – contains Eclipse hardware and software Launch files for both devices (relevant only when developing for JN5148 in the Eclipse IDE)

The **AN1046\_154\_Coord** and **AN1046\_154\_EndD** folders each contain **Source** and **Build** sub-folders, the contents of which are described below.

#### Source Folders

The contents of the **Source** folders are as follows:

- **AN1046\_154\_Coord/Source** contains the file **AN1046\_154\_Coord.c** which contains the source code for the PAN Co-ordinator.
- **AN1046\_154\_EndD/Source** contains the file **AN1046\_154\_EndD.c** which contains the source code for an End Device.

To adapt the skeleton code to your own needs, you may need to modify the above source files.

## Build Folders

The contents of the **Build** folders are similar for the two devices, comprising the following makefiles:

- **Makefile**: This is the makefile for command line compilation of the source code for the JN5148 wireless microcontroller.
- **Makefile\_JN5139.mk**: This is the makefile for command line compilation of the source code for the JN5139 or JN5121 wireless microcontroller.

You will only need the above makefiles if you choose to develop applications in a command line environment.

The **Build** folder is also the place where a makefile compilation or Eclipse compilation outputs the resulting binary file.



**Note:** When an application is built in Code::Blocks, the resulting binary file is output to a folder **JN5139\_Build** or **JN5121\_Build**, depending on the target device. This folder is created during the build process.

---

## 3.2 Code Descriptions

This section describes the supplied source code at function level. The sub-sections below describe the code for the PAN Co-ordinator and the code for the End Device.

The **config.h** header file is referenced in both source files, as are the following Jennic header files: **jendefs.h**, **AppHardwareApi.h**, **AppQueueApi.h**, **mac\_sap.h** and **mac\_pib.h**.

---

### 3.2.1 Contents of AN1046\_154\_Coord.c

The entry point from the boot loader into the Co-ordinator application is the function **AppColdStart()** – this is the equivalent of the **main()** function in other C programs. This function performs the following tasks (also illustrated in Figure 4):

1. **AppColdStart()** calls the function **vInitSystem()**, which itself performs the following tasks:
  - Initialises the IEEE 802.15.4 stack on the device
  - Sets the PAN ID and short address of the PAN Co-ordinator – in this application, these are pre-determined and are defined in the file **config.h**
  - Turns on the radio receiver
  - Enables the device to accept association requests from other devices
2. **AppColdStart()** calls the function **vStartEnergyScan()** which starts an Energy Detection Scan to assess the level of activity in the possible radio frequency channels – the channels to be scanned are defined in the file **config.h** along with the scan duration. Initiation of the scan is handled as an MLME request to the IEEE 802.15.4 MAC sub-layer.
3. **AppColdStart()** waits for an MLME response using the function **vProcessEventQueues()** – this function checks each of the three event queues and processes items found. The function uses the function **vProcessIncomingMlme()** to handle the MLME response. This function calls **vHandleEnergyScanResponse()** which processes the results of the Energy Detection Scan – the function searches the results to find the quietest channel and sets this as the adopted channel for the network. The latter function then calls **vStartCoordinator()** which sets the required parameters and then submits an MLME request to start the network – note that no response is expected for this request.
4. **AppColdStart()** loops the function **vProcessEventQueues()** to wait for an association request from another device, which arrives as an MLME request (note that the beacon request from the device is handled by the IEEE 802.15.4 stack and is not seen by the application). When the association request arrives, the function **vHandleNodeAssociation()** is called to process the request. The latter function creates and submits an association response via MLME.
5. **AppColdStart()** loops the function **vProcessEventQueues()** to wait for messages from the associated device arriving via the MCPS and hardware queues.
  - When data arrives in the MCPS queue, **vProcessEventQueues()** first uses the function **vProcessIncomingMcps()** to accept the incoming data frame. Note that **vProcessIncomingMcps()** uses **vHandleMcpsDataInd()**, which calls **vProcessReceivedDataPacket()** in which you must define the processing to be done on the data.
  - When an event arrives in the hardware queue, **vProcessEventQueues()** calls the function **vProcessIncomingHwEvent()** to accept the incoming event. You must define the processing to be performed in this function.



**Note:** As it stands, the code is only designed to receive data. To transmit data from the PAN Co-ordinator, you must modify the code – a transmission function is provided (see Section 3.3.6).

The above Co-ordinator set-up process is illustrated in the figure below.

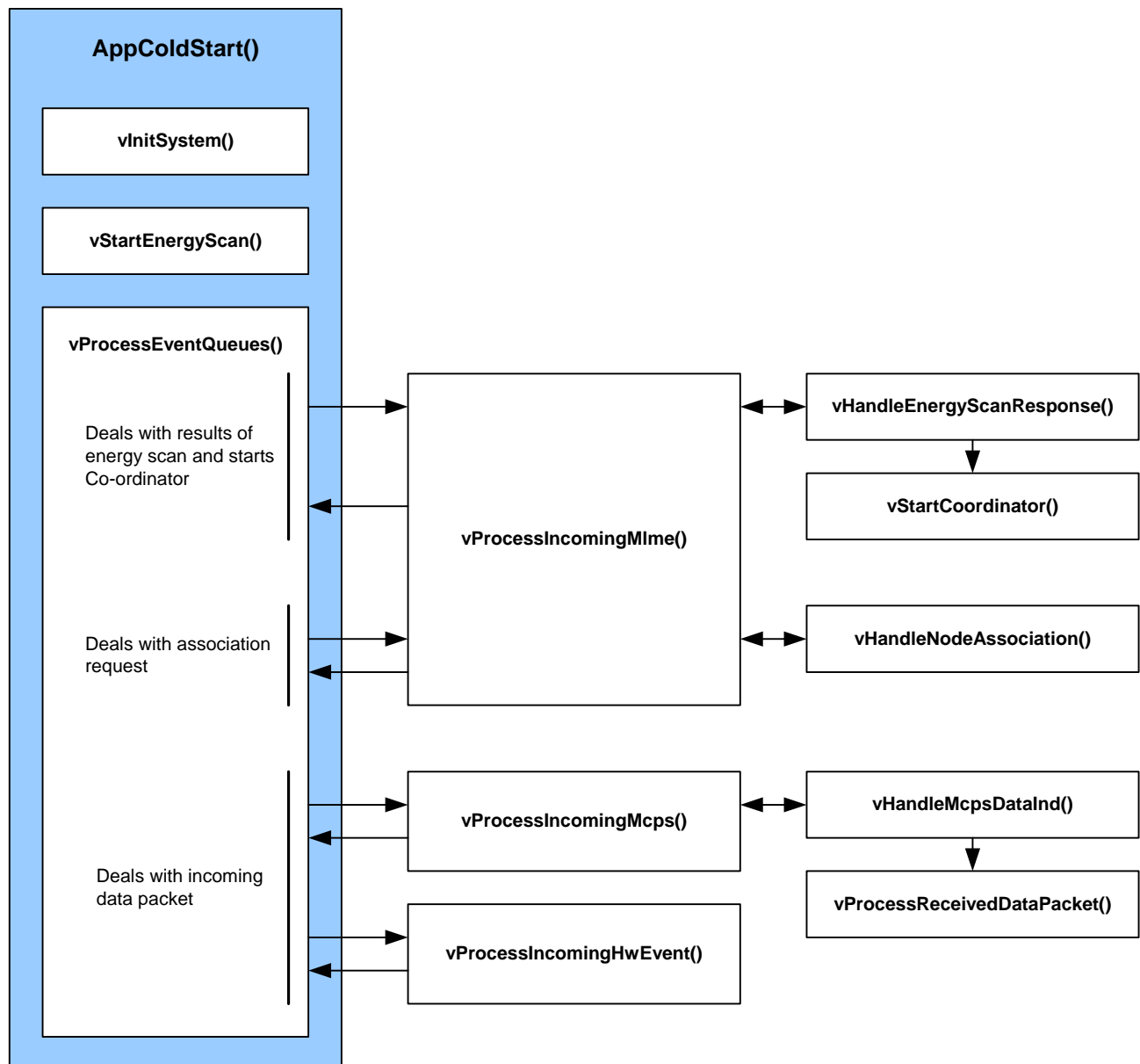


Figure 4: PAN Co-ordinator Set-up Process

---

### 3.2.2 Contents of AN1046\_154\_EndD.c

The entry point from the boot loader into the End Device application is the function **AppColdStart()** – this is the equivalent of the **main()** function in other C programs. In **AN1046\_154\_EndD.c**, this function is defined differently from that in **AN1046\_154\_Coord.c**. For the End Device, it performs the following tasks (also illustrated in Figure 5):

1. **AppColdStart()** calls the function **vInitSystem()**, which initialises the IEEE 802.15.4 stack on the device.
2. **AppColdStart()** calls the function **vStartActiveScan()** which starts an Active Channel Scan in which the device sends beacon requests to be detected by the PAN Co-ordinator, which then sends out a beacon in response – the channels to be scanned are defined in the file **config.h** along with the scan duration. Initiation of the scan is handled as an MLME request to the IEEE 802.15.4 MAC sub-layer.
3. **AppColdStart()** waits for an MLME response using the function **vProcessEventQueues()** which checks each of the three event queues and processes the items it finds. The function uses the **vProcessIncomingMlme()** function to handle the MLME response. This function calls the function **vHandleActiveScanResponse()** which processes the results of the Active Channel Scan:
  - If a PAN Co-ordinator is found, the function stores the Co-ordinator details (PAN ID, short address, logical channel) and calls **vStartAssociate()** to submit an association request to the Co-ordinator – this is handled as an MLME request.
  - If a PAN Co-ordinator is not found (possibly because the Co-ordinator has not yet been initialised), the function recalls **vStartActiveScan()** in order to restart the scan (in which case this process continues as described from Step 2).
4. **AppColdStart()** loops the function **vProcessEventQueues()** to wait for an association response from the Co-ordinator. When the response is received, **vProcessIncomingMlme()** is called, which (provided that the device is in the associating state) calls the function **vHandleAssociateResponse()** to process the response. The latter function checks the association response:
  - If the PAN Co-ordinator has accepted the association, the function puts the device in the ‘associated’ state.
  - If the PAN Co-ordinator has rejected the association, the function recalls **vStartActiveScan()** to start a search for another PAN Co-ordinator (in which case this process continues as described from Step 2).

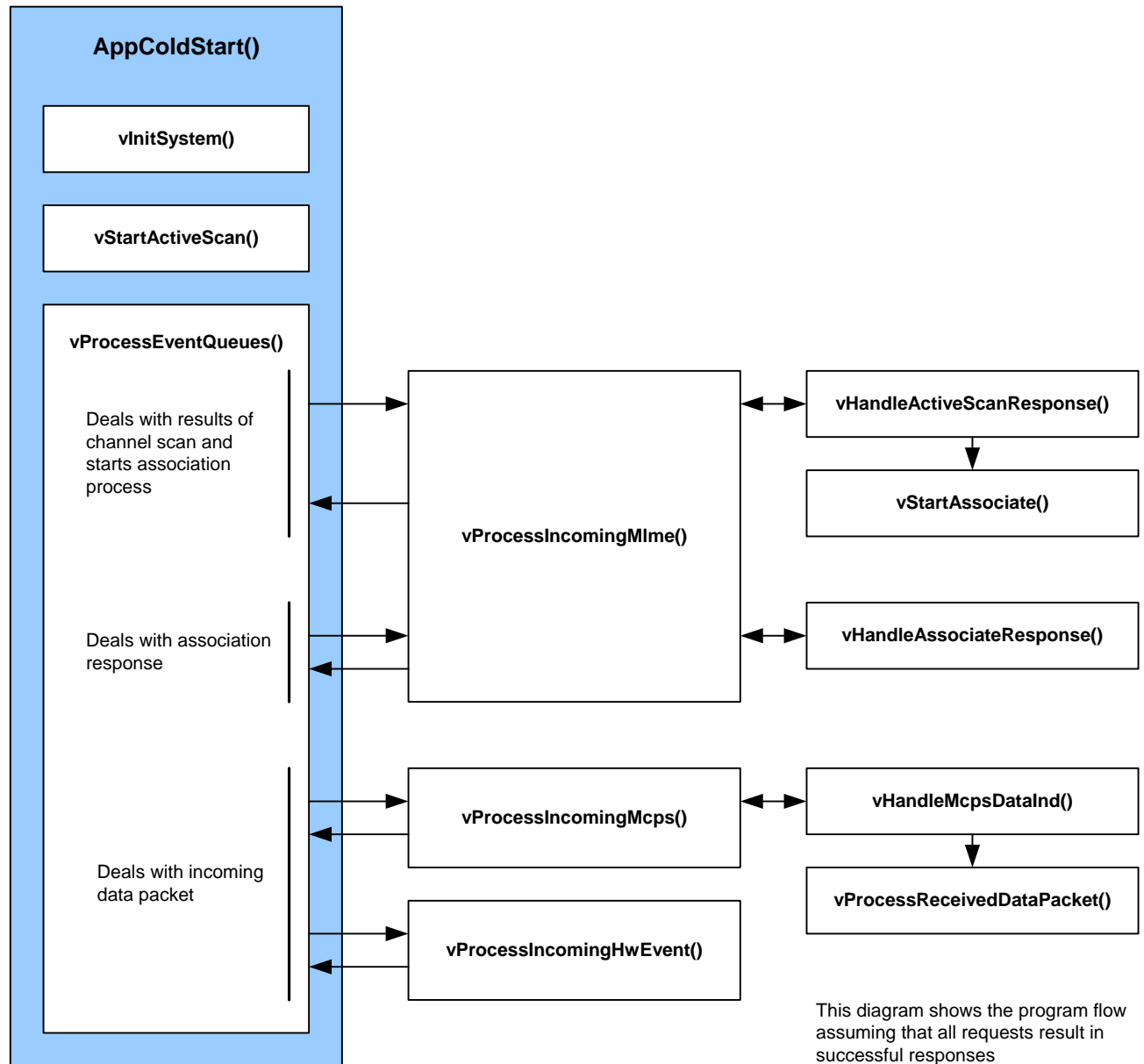
5. **AppColdStart()** loops the function **vProcessEventQueues()** to wait for messages from the PAN Co-ordinator arriving via the MCPS and hardware queues.

- When data arrives in the MCPS queue, **vProcessEventQueues()** first uses the function **vProcessIncomingMcps()** to accept the incoming data frame. Note that **vProcessIncomingMcps()** uses **vHandleMcpsDataInd()**, which calls **vProcessReceivedDataPacket()** in which you must define the processing to be done on the data.
- When an event arrives in the hardware queue, **vProcessEventQueues()** calls the function **vProcessIncomingHwEvent()** to accept the incoming event. You must define the processing to be performed in this function.



**Note:** As it stands, the code is only designed to receive data. To transmit data from the device, you must modify the code – a transmission function is provided (see Section 3.3.6).

The above End Device set-up process is illustrated in the figure below.

**Figure 5: End Device Set-up Process**



---

## 3.3 Adapting the Skeleton Code

This section provides guidelines on how to modify the supplied skeleton code to achieve different requirements. The modifications covered are:

- How Do I Program a Pre-defined PAN ID?
- How Do I Program Pre-defined Short Addresses?
- How Do I Add End Devices to the Network?
- How Do I Program the Channel Scans?
- How Do I Define the Processing of Received Data Packets?
- How Do I Program Data Transmission?

---

### 3.3.1 How Do I Program a Pre-defined PAN ID?

The PAN ID is pre-defined in the file **config.h**. In the skeleton code, it is set to 0xCAFE.

To use a different PAN ID, open **config.h** and change the hex number in the following line:

```
#define PAN_ID                                0xCAFE
```



**Caution:** The chosen PAN ID must not conflict with the PAN IDs of any other IEEE 802.15.4 based networks in the vicinity.

---

### 3.3.2 How Do I Program Pre-defined Short Addresses?

The 16-bit short addresses of the PAN Co-ordinator and End Device are pre-defined in the file **config.h**. In the skeleton code, the short addresses are set to 0x0000 for the Co-ordinator and 0x0001 for the first End Device. The latter is a start address for the End Devices – if you have multiple End Devices, their short addresses will be automatically numbered from this value upwards in increments of 0x0001.

To use different short addresses, open **config.h** and change the hex numbers in the following lines:

```
#define COORDINATOR_ADR                      0x0000  
#define END_DEVICE_START_ADR                0x0001
```



**Note:** It is usual to set 0x0000 as the short address of the PAN Co-ordinator.

---

### 3.3.3 How Do I Add End Devices to the Network?

The skeleton code is designed for a network consisting of at least two devices; a PAN Co-ordinator and an End Device. By default, the maximum number of End Devices defined in the code is 10 – this means that you can use up to ten End Devices without any modifications. However, you can use more End Devices by modifying the code as described below.



**Note:** When using multiple End Devices, their short addresses are automatically assigned starting with the address `END_DEVICE_START_ADR` defined in the **config.h** file – see Section 3.3.2.

#### Modifications to config.h

The file **config.h** contains a line defining the maximum number of End Devices supported by the application – in the supplied code, it is set to 10, as shown below:

```
#define MAX_END_DEVICES 10
```

To increase or decrease the maximum number of End Devices, open **config.h** and change this number.

#### Modifications to AN1046\_154\_EndD.c

The source file **AN1046\_154\_EndD.c** provides the code to be loaded in an End Device. If you have more than one End Device, and they are of different types (e.g. one a temperature sensor, the other a humidity sensor), they are likely to need different source code. Therefore, when adding End Devices, you may need to devise specific code for the new devices.

#### Modifications to AN1046\_154\_Coord.c

To add End Devices to your network, you do not need to modify the file **AN1046\_154\_Coord.c**.

---

### 3.3.4 How Do I Program the Channel Scans?

The skeleton code involves two frequency channel scans:

- An Energy Detection Scan invoked by the PAN Co-ordinator during network set-up to find the most suitable channel for network operation.
- An Active Channel Scan invoked by the End Device during device association to find the operating channel of the PAN Co-ordinator.

It is not normally necessary to check all possible frequency channels. The 27 channels (numbered 0 to 26) of the IEEE 802.15.4 standard are distributed among the three frequency bands (868, 915 and 2400 MHz). Since a network is usually intended to work in only one of these bands, there is little point in scanning channels in the other two bands (Jennic products operate in the 2400-MHz band; channels 11

to 26). In addition, you may be aware that another network in the locality already operates in one of the channels, so this channel should be excluded from the scan. Therefore, you can pre-define the channels that will be checked in these scans. You can also define the amount of time spent checking each channel in each of the scans. These definitions are made in the header file **config.h**, as described below.

### Defining the Channels to be Scanned

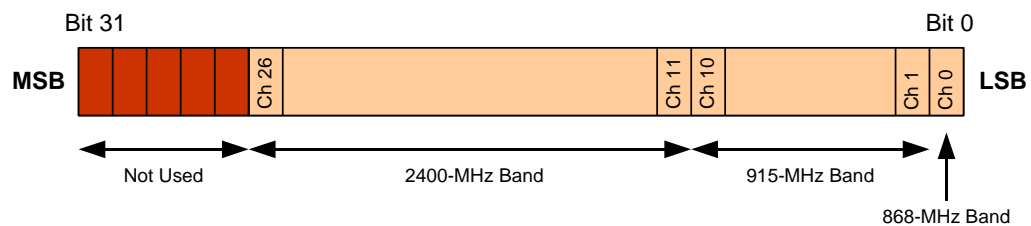
The file **config.h** includes the following line:

```
#define SCAN_CHANNELS 0x07FFF800UL
```

SCAN\_CHANNELS defines exactly which channels will be scanned. Each bit of the value (0x07FFF800 in this case) corresponds to a channel, where the least significant bit (LSB) corresponds to channel 0; see Figure 6.

- A bit value of 1 means 'scan'.
- A bit value of 0 means 'do not scan'.

To change the channels to be scanned, modify this hex value.



**Figure 6: Channel Allocations in SCAN\_CHANNELS**



**Note:** SCAN\_CHANNELS applies to both the Energy Detection Scan and the Active Channel Scan.



**Caution:** Since the JN51xx wireless microcontroller only operates in the 2400-MHz band, there is no point in configuring scans in channels of the lower bands.

### Defining the Channel Scan Durations

The file **config.h** includes the following two lines:

```
#define ACTIVE_SCAN_DURATION      3
#define ENERGY_SCAN_DURATION    3
```

Each of these parameters defines the time taken to check each channel in a scan:

- ACTIVE\_SCAN\_DURATION for an Active Channel Scan
- ENERGY\_SCAN\_DURATION for an Energy Detection Scan

These parameters take a positive integer value that determines the scan duration per channel, in milliseconds, according to the following formulae:

**For an Active Channel Scan:**

$$\text{Channel scan duration (ms)} = 15.36 \times (2^{\text{ACTIVE\_SCAN\_DURATION}} + 1)$$

**For an Energy Detection Scan:**

$$\text{Channel scan duration (ms)} = 15.36 \times (2^{\text{ENERGY\_SCAN\_DURATION}} + 1)$$

Thus, in each case, a value of 3 gives a channel scan duration of 138.24 ms.

To change the channel scan durations, modify the above code values.



**Note:** The value of each of ACTIVE\_SCAN\_DURATION and ENERGY\_SCAN\_DURATION must be an integer in the range 0 to 14 (inclusive). Thus, the channel scan durations can be in the range 30.72 ms to 251.6736 s.

### 3.3.5 How Do I Define the Processing of Received Data Packets?

The IEEE 802.15.4 stack puts an incoming data packet into the MCPS queue on the destination device. The skeleton code for both the PAN Co-ordinator and End Device will retrieve the data packet from the queue but will not process the data in any way – you must define how you want to process the data. However, an empty function already exists in the code to accommodate your data processing code - **vProcessReceivedDataPacket()**. You must define the required processing for this function in the files **AN1046\_154\_Coord.c** and **AN1046\_154\_EndD.c**.



**Note:** The empty function **vProcessReceivedDataPacket()** appears in both **AN1046\_154\_Coord.c** and **AN1046\_154\_EndD.c**. However, the PAN Co-ordinator is likely to process received data packets in a different way from an End Device. Therefore, you are likely to define **vProcessReceivedDataPacket()** differently in the two source files.

---

### 3.3.6 How Do I Program Data Transmission?

In each of the source files **AN1046\_154\_Coord.c** and **AN1046\_154\_EndD.c**, a function for transmitting data is already defined - **vTransmitDataPacket()**. You simply need to add code to call this function as appropriate for your application.



---

## 4 Compiling Your Application Code

This chapter describes how to build your code and then download it to the network devices.

---

### 4.1 Building Your Code

Once you have modified the source files **AN1046\_154\_Coord.c** and **AN1046\_154\_EndD.c** (as well as the header file **config.h**) according to your needs, you must build the executables on a PC or workstation before downloading them to the relevant network devices.

The available build methods depend on the chip type (JN5148, JN5139 or JN5121):

- JN5148 applications can be built using the Eclipse IDE or makefiles
- JN5139 and JN5121 applications can be built using the Code::Blocks IDE or makefiles

Build the applications as described in the appropriate section below, depending on whether you intend to use Eclipse, Code::Blocks or makefiles.

---

#### 4.1.1 Building Code Using Makefiles

This section describes how to build your application code using the makefiles supplied in the **Build** folder for each application (see Section 3.1.3). For both the PAN Co-ordinator and End Device applications, the makefiles are named as follows:

- **Makefile**: This is the makefile for a JN5148 chip.
- **Makefile\_JN5139.mk**: This is the makefile for a JN5139 or JN5121 chip.

To build each application and load it into a JN51xx board, follow the instructions below:

1. Ensure that the project directory is located in

**<JENNIC\_SDK\_ROOT>\Application** for JN5148

**<JENNIC\_SDK\_ROOT>\cygwin\jennic\SDK\Application** for JN5139/JN5121

where **<JENNIC\_SDK\_ROOT>** is the path into which the Jennic SDK was installed.

2. Navigate to the **Build** directory for the application to be built and follow the instructions below for your chip type:

#### **For JN5148:**

At the command prompt, enter:

```
make clean all
```

Note that for the JN5148, you can alternatively enter the above command from the top level of the project directory, which will build the binaries for both the applications.

**For JN5139:**

At the command prompt, enter:

```
make -f Makefile_JN5139.mk clean all
```

**For JN5121:**

At the command prompt, enter:

```
make -f Makefile_JN5139.mk JENNIC_CHIP=JN5121 clean all
```


In all the above cases, the binary file will be created in the **Build** directory, the resulting filename reflecting the name of the source file and the chip type (JN5148, JN5139 or JN5121) for which the application has been built.

3. Load the resulting binary files from the **Build** directory into the boards. To do this, use the Jennic JN51xx Flash Programmer, described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*.

---

### 4.1.2 Building Code Using Eclipse (JN5148 Only)

To build the application and load it into JN5148 boards, follow the instructions below:

1. Ensure that the project directory is located in  
**<JENNIC\_SDK\_ROOT>\Application**  
where **<JENNIC\_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**, select **Existing Projects into Workspace** and click **Next**.
  - c) Enable **Select root directory**, browse to the Jennic **Application** directory and click **OK**.
  - 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 Eclipse and use the drop-down list associated with the hammer icon  in the Eclipse toolbar to select the relevant build configuration – once selected, the application will automatically build. Repeat this to build the other application.

The binary files will be created in the relevant **Build** directory.

4. Load the resulting binary files into the boards. Do this using the Jennic JN51xx Flash Programmer, which can be launched from within Eclipse or used directly (and is described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*).



### 4.1.3 Building Code Using Code::Blocks (JN5139/JN5121 Only)

The Code::Blocks project files are located in the directory

**..\SDK\Application\<Project>\CodeBlocksProject**

A project file is provided for each device type (Co-ordinator, End Device), for each type of Jennic microcontroller (your chip type is marked on the chip or module).

For the JN5139 wireless microcontroller, the files are:

**AN1046\_154\_Coord\_JN5139.cbp**  
**AN1046\_154\_EndD\_JN5139.cbp**

For the JN5121 wireless microcontroller, the files are:

**AN1046\_154\_Coord\_JN5121.cbp**  
**AN1046\_154\_EndD\_JN5121.cbp**




**Caution:** You must use the version of Code::Blocks that is available from the Support area of the Jennic web site ([www.jennic.com/support](http://www.jennic.com/support)).

To build each application and load it into a JN5139 or JN5121 board, follow the instructions below:

1. Ensure that the project directory is located in

**<JENNIC\_SDK\_ROOT>\cygwin\jennic\SDK\Application**

where **<JENNIC\_SDK\_ROOT>** is the path into which the Jennic SDK was installed.

2. Open the appropriate Code::Blocks project file (**.cbp** file in the **CodeBlocksProject** directory) and build using the Build button  in the toolbar.

The binary file will be created in the **5139\_Build** or **5121\_Build** directory, the resulting filename matching that of the project file used to create it.

3. Load the resulting binary file into the board. Do this using the Jennic JN51xx Flash Programmer, which can be launched from within Code::Blocks or used directly (and is described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*).



## Revision History

Version	Date	Description
1.0	21-Sep-2006	Initial release
1.1	15-Nov-2006	Updated to refer to Application Note JN-AN-1046 which now contains the application template (the template was previously bundled with this manual). Added references to Code::Blocks IDE on the Jennic web site.
1.2	16-Apr-2007	Updated for the JN513x series chips and new SDK structure
1.3	24-May-2007	Updated for R1 of JN5139 device
1.4	01-Feb-2008	Updated for JN5139 and new SDK
1.5	02-Oct-2008	Note about data broadcasts added to end of Chapter 2
2.0	18-Jun-2010	Updated for JN5148 and associated SDK

## 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.

**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](mailto:info@jennic.com)

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

**www.Jennic.com**  
TECHNOLOGY FOR A CHANGING WORLD