Application Note: JN-AN-1222

ZigBee IoT Gateway with NFC

This Application Note describes the hardware and software components that are required to implement the Host processor part of a Linux-based ZigBee 'Internet of Things' (IoT) Gateway using NXP's Near Field Communication (NFC) feature.

Creating a ZigBee IoT Gateway based on Linux allows the platform to utilize the vast array of software available for the operating system to create a highly featured, robust and scalable solution with a fast time-to-market.

1 Application Overview

The Linux-based ZigBee IoT Gateway from NXP allows the connection of a ZigBee network to the Internet via an application data translation process in which ZigBee devices can exchange data with any other connected device, enabling the 'Internet of Things'. This document describes the software on the host processor, which forms the 'ZigBee IoT Gateway Host'. It must be interfaced with a ZigBee Control Bridge (ZCB) in order to form a complete ZigBee IoT Gateway. This is described in the accompanying Application Notes *ZigBee IoT Control Bridge (JN-AN-1223)* and *ZigBee 3.0 IoT Control Bridge (JN-AN-1216)*. Moreover, the complete system is described in the *IoT Gateway User Guide (JN-UG-3117)*, included in this Application Note package*.*

1.1 Hardware

The hardware required to create the system described in this Application Note can be taken from either the JN516x-EK004 or JN517x-DK005 hardware kit.

The following parts from either of the above kits are required to build the gateway:

- Raspberry Pi board, plus power adapter
- Micro-SD Card
- WiPi Wi-Fi USB dongle or Ethernet cable for connection to an existing LAN
- JN5169 USB dongle (OM15020) or JN5179 USB dongle (OM15021)
- Extension board for NFC reader (OM5577/PN7120S)

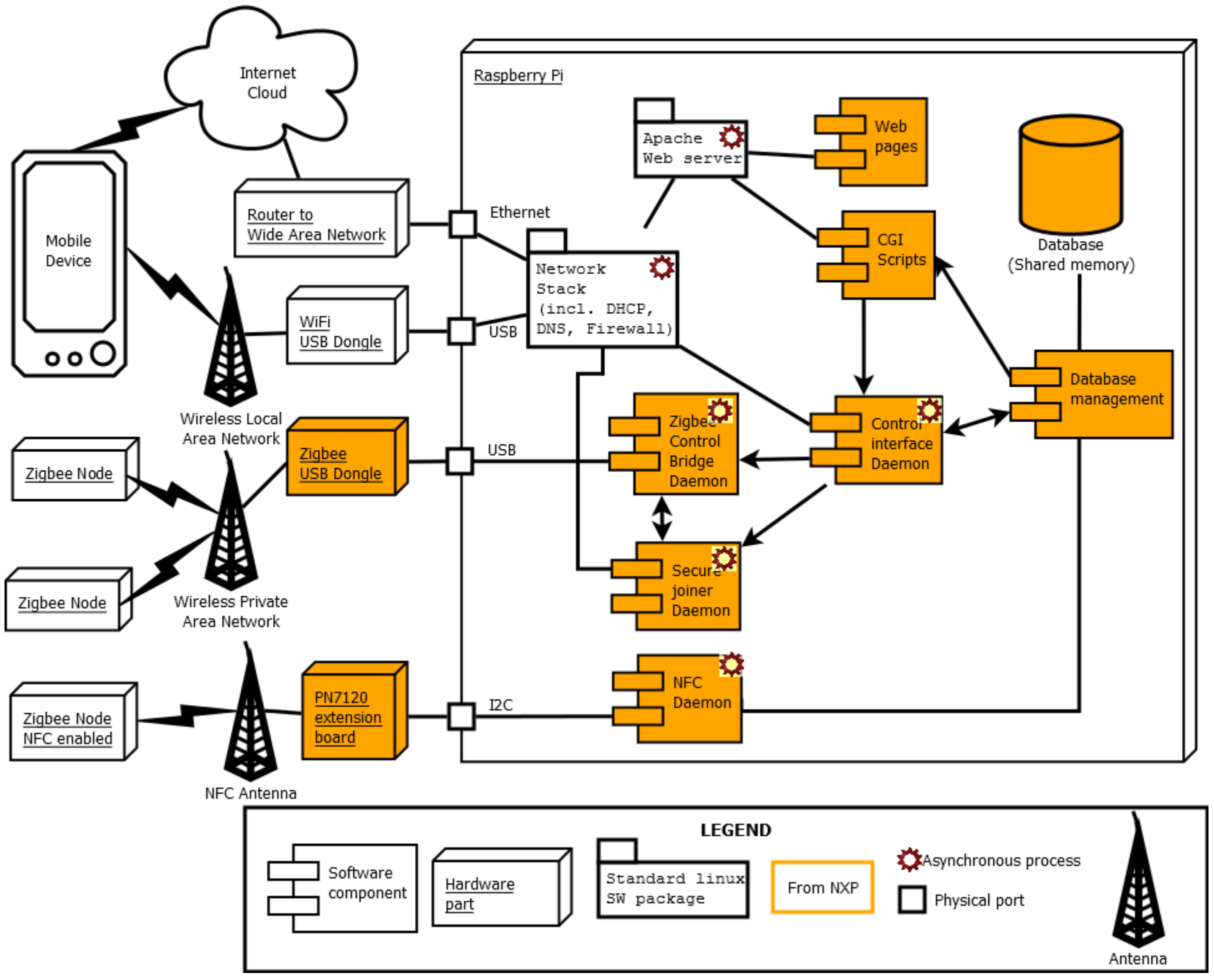
1.2 Operating System

The NXP IoT Gateway host functionality can be added to any Linux system, using common system packages and a few developed by NXP. The firmware used in the NXP IoT Gateway is compatible with the JN516x-EK004 and JN517x-DK005 hardware kits. This firmware is based upon the OpenWrt Linux distribution. This distribution was chosen because it has support for many cheap commercial off-the-shelf WiFi routers, and is also easy to port to new hardware platforms. It is highly configurable and has a wide variety of packages available for installation, providing a solution to almost any networking requirement.

2 Software Architecture

2.1 Overview

The overview depicted below shows the essential IoT daemons and the dataflow between them. A description of each system component follows in the next sections.



2.2 Main Components

2.2.1 IoT Database

The IoT database forms the heart of the IoT Gateway. It contains the following tables:

Table	Description
System	Contains system settings, e.g. current ZigBee Channel
Devices	Contains all devices that are currently in the network.
Plug history	Keeps a 24-hour plug meter history for generating usage overviews
ZCB	Low-level mapping table between ZigBee's short (network) addresses and extended addresses

2.2.2 ZigBee Control Bridge (ZCB)

The ZigBee Control Bridge runs the ZigBee HA software and sends/receives fully compliant ZigBee HA messages/clusters. This control bridge is the Coordinator in the ZigBee network. The software runs on a JN516x or JN517x USB Dongle, and communication with this device from Linux world is done over the serial port /dev/ttyUSB0.

The communication protocol is described in the Application Notes *ZigBee IoT Gateway Control Bridge (JN-AN-1223)* and *ZigBee 3.0 IoT Control Bridge (JN-AN-1216)*.

2.2.3 ZCB Daemon

The ZCB daemon is the counterpart of the ZCB USB dongle software. It is called iot_zb and can be started with following syntax: iot_zb

No parameters are needed. This program communicates with the other IoT programs over two message queues:

- An input queue for IoT commands to the ZCB (e.g. set channel) or into the ZigBee network (e.g. controlling the color of a lamp),
- An output queue towards the Dispatcher (e.g. a new energy consumption reading from a plug meter) where further processing or relaying takes place.

Messages over these IoT queues are JSON encoded, human-readable messages. The queue system makes sure that messages from different sources do not get intermixed.

Note: The ZCB daemon also has a supporting function towards the ZCB in the sense that it keeps a translation table from ZigBee's short (network) addresses to extended addresses. Allocating this table into the IoT Gateway saves valuable (RAM) memory space in the ZCB so that it can support up to 250 nodes.

2.2.4 Secure Joiner

The Secure Joiner program is called "iot_sj" and can be started with following syntax:

```
iot_sj
```

No parameters are needed. This program is required in the Single-Touch Commissioning scenario 3 described in the *NFC Commissioning User Guide (JN-UG-3112)*. It serves a socket connection through which clients can ask for the encrypted network key (secjoin structure) given their device-specific linkinfo structure. The commissioning process is further explained in Section 2.3, NFC Commissioning.

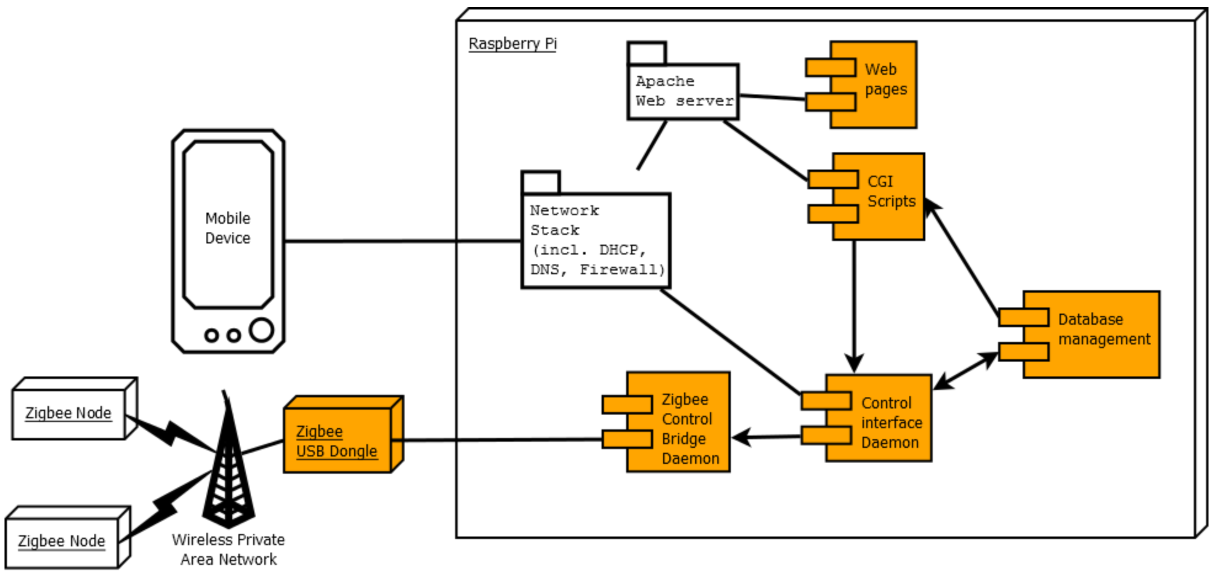
2.2.5 Control Interface

The Control Interface program is called "iot_ci" and can be started with following syntax:

```
iot_ci
```

No parameters are needed.

The diagram below gives an overview on how the Control Interface interacts in the system.



This program handles all other non-secure-join remote accesses to the system via a socket port (2001), setting the topology by the web interface, database interrogations, set point commands and system controls.

The dataflow of commands towards the ZCB, and in some cases further into the wireless network, goes via the input queue of the ZCB-Linux. Responses go back to the Control Interface queue.

2.2.6 NFC Daemon

The NFC daemon program is called “iot_nd” and can be started with following syntax:

```
iot_nd
```

No parameters are needed. The NFC daemon implements the NFC reader functionality and polls for NFC-forum Type2 tags with NDEF formatted payloads. Based on the URL in the NDEF header, the corresponding “NDEF-app” is called with the rest of the NDEF payload.

In order for the PN7120-based NFC reader software to work, the Linux OS must provide the following user-space functionality:

- Device “/dev/i2c-1” must be present in order to be able to communicate with the PN7120
- Two GPIO pins must be accessible through the sys-fs file system /sys/class/gpio in order to enable the PN7120 and react to its hardware interrupt

2.2.7 CGI Scripts

The IoT CGI scripts are invoked by the web server to generate web pages to external web clients (e.g. PCs or mobile devices). The CGI scripts are written in ‘C’ and can access the IoT database directly (for reading only) and communicate with the other IoT programs through sockets and queues, wherever applicable.

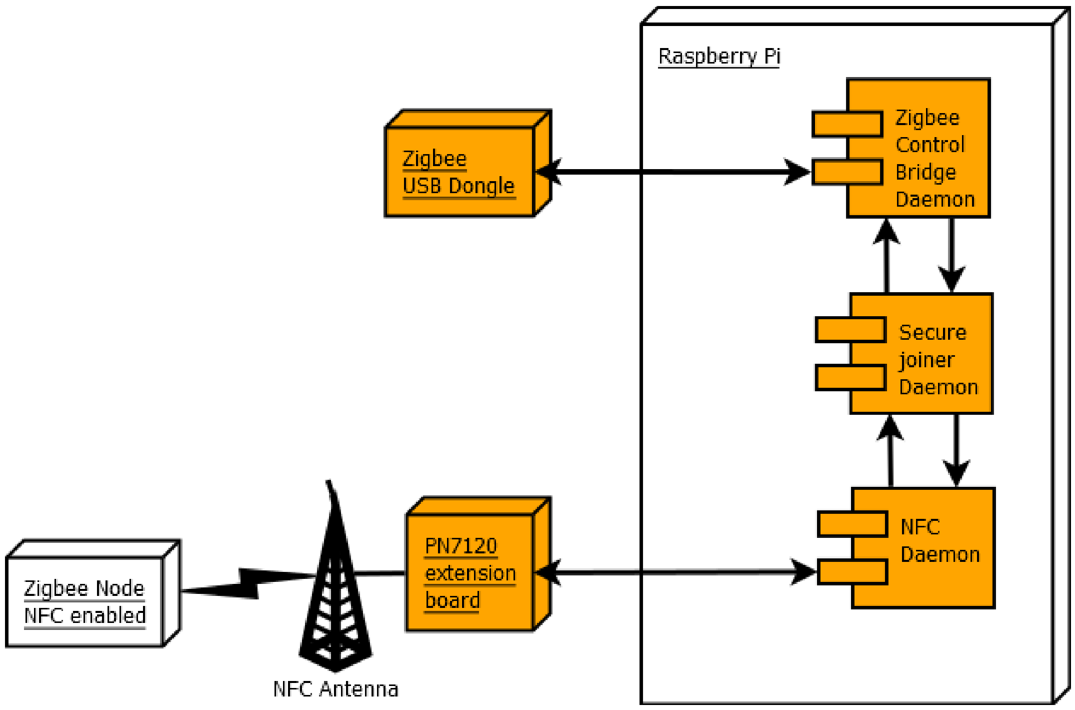
Most CGI scripts work in close cooperation with an accompanying JavaScript program running on the web client. Underlying HTML5 AJAX technology is used to sync and seamlessly update the web pages’ contents with the actual data on the IoT Gateway.

2.3 NFC Commissioning

The Secure Joiner program plays a key role in the (secure) NFC commissioning process. There are two potential clients for the Secure Joiner: an NFC enabled mobile phone running a commissioning app (not provided in the evaluation kit) or the Gateway's NFC reader.

2.3.1 Commissioning Nodes into the Network via the NFC Reader

When using the Gateway's NFC reader, the commissioning/decommissioning mode can be set from the web interface, prior to touching a device. Let us assume the NFC reader daemon is in commissioning mode.



After an IoT-compatible device has touched the Gateway's NFC reader for single-touch commissioning, the flow through the system goes as follows:

1. The NFC daemon registers at the secure join socket (port 2000).
2. The NFC daemon sends an *linkinfo* data structure which it gets from the device that is currently being single-touch commissioned.
3. The Secure Joiner reads this socket message, translates it into an "authorization request" message and sends it to the ZCB via the input queue of the ZCB daemon.
4. The ZCB processes the data and responds with a ZCB "authorize answer" that is picked up by the ZCB daemon and transferred to the Dispatcher daemon via its message queue.
5. The Dispatcher daemon sends the ZCB message to the Secure Joiner daemon via its message queue.
6. The Secure Joiner daemon translates this ZCB "authorize" message into a secjoin message and sends it to the NFC reader over the socket.
7. The NFC daemon writes the result back into the device, which can then be commissioned into the wireless network.

If the system cannot handle the *linkinfo* data structure, no response is sent, a timeout occurs and a corresponding error message is sent to the NFC reader. During the commissioning and decommissioning actions, the NFC reader gives audible and LED feedback to the user.

Note: The Installation Code NFC commissioning has also been implemented to take benefit from the ZigBee 3.0 out of band commissioning. If the ZigBee end node supports the installation code, the gateway replaces the *linkinfo* structure with *oob_request_info* structure to match the NTAG data type.

2.3.2 Secure Commissioning

To prevent the exposure of critical ZigBee information in the NTAG-I2C, it is possible to apply a public key exchange (ECDH scheme) to derive an AES session key and encrypt the ZigBee information stored in the NTAG-I2C. Therefore, the Host software is already prepared to run secure commissioning and decommissioning, although this is not used in the JN516x-EK004 or JN517x-DK005 kit.

2.3.3 Decommissioning with NFC Reader

The data flow for decommissioning is much easier. When the device touches the NFC antenna, the reader just writes the decommission command into the NTAG-I2C tag, after which the node can process the command and leave the network. Decommissioning also affects the device database, as it is directly updated in the web interface even when the device is unpowered.

3 OpenWrt

The OpenWrt Linux distribution was chosen due to its large hardware support, small footprint, ease of use and large set of available packages. OpenWrt uses a heavily modified Buildroot environment to control the build process.

The following sections explain:

- How to build a complete file system image containing both the operating system and the Gateway host software.
- How to modify the Gateway source code and recompile it.

Warnings:

- Do everything as a non-root user!
- Issue all OpenWrt Buildroot commands in the <buildsystem root> directory, e.g. ~/openWrt/
- Do not build in a directory that has spaces in its full path

3.1 Building File System Image

3.1.1 Prerequisites

OpenWrt Buildroot is the build system for the OpenWrt Linux distribution. OpenWrt Buildroot works on Linux. A case-sensitive file system is required and, therefore, it is not possible to use a Cygwin Linux emulation system.

It is recommended to use a Linux distribution (Debian based), either a standalone installation or one running in a virtual environment.

If a virtual environment is chosen, please apply the following properties for the virtual machine:

- PC with x86 compatible CPU processor
- 3GB of RAM
- HDD capacity of 30GB

For more information, please refer to the OpenWrt web site (1) (2).

Note: Since most of the required software packages are downloaded from remote servers, Internet access is mandatory for the following protocols: https, ftp, git, svn.

3.1.2 Setting up the Build Environment

Once the Linux system is ready, a number of software packages need to be installed.

OpenWrt buildroot requirements

Package git is required to conveniently download the OpenWrt source code and build tools to do the cross-compilation process. Open a Linux terminal and enter the following commands:

```
sudo apt-get update

sudo apt-get install git-core build-essential libssl-dev
libncurses5-dev unzip wget
```

Some feeds might not be available over git but only via subversion (svn) or mercurial. To obtain these source-codes, svn and mercurial must be installed as well, using the command below:

```
sudo apt-get install subversion mercurial
```

Application toolchain requirements

Gateway software components need additional packages that are provided with the following command:

```
sudo apt-get install gawk zlib1g libncurses5 g++ flex
```

3.1.3 Building the File System Image

Now that the build environment is ready, you need to decompress the source code archive. In this document, we will assume that the source code archive is located in your home directory. After decompression of the archive, go to directory ~/JN–AN–1222/Source/Host/openWrt and start the build process with the following command:

make TARGET=*rpi_version*

rpi_version refers to the version of the Raspberry Pi board that is included into your JN516x/JN517x hardware kit. The board version is written on the PCB of the Raspberry Pi. Currently, the Raspberry Pi 2 Model B V1.1 is the version officially supported, and the Raspberry Pi Model B+ has become unsupported (obsolete).

The table below gives the value of *rpi_version* for each board version.

Raspberry Pi Board's Version	***rpi_version* to use**
Raspberry Pi Model B+ V1.2 **(obsolete)**	RPi-B+
Raspberry Pi 2 Model B V1.1	RPi2-B

The build process is made of several steps:

1. Download a known and stable version of the OpenWrt operating system.
2. Download all the software packages required to make a complete system.
3. Apply local patches where needed.
4. Perform auto-configuration of the buildroot to match the JN516x/JN517x kit hardware.
5. Build host and target development toolchains (compilers, linkers, utilities, etc).
6. Compile and build each software package.
7. Finally assemble the complete file system image.

This compilation step can take several hours. So if a standalone Linux system with multiple CPU cores is used, the compilation process will be faster with the option –j X (replace X by the number of cores), as shown below for 2 cores:

```
make –j 2 TARGET=rpi_version
```

Note: On some Linux systems, the build process may fail if the path from which the build is launched is too long. If this problem occurs, you will need to reduce the length of buildroot's path. One way to do this is to move the openWrt directory to your home directory and build from that location, like in the example below:

```
cp –R ~/JN-AN-1222/Source/Host/openWrt ~/myBuildRoot
cd ~/myBuildroot
make TARGET=rpi_version
```

3.1.4 Cleaning buildroot

If you want to clean everything, use the following command:

```
make clean_all
```

Note: This command will remove the complete build-root, including all the software packages that have been downloaded. We recommend that these software packages are saved beforehand with the following command (to be adapted according to your environment, xx.yy represents the version of the OpenWrt system):

```
cp –R ~/JN-AN-1222/Source/Host/openWrt/openWrt-xx.yy/dl ~/openWrtFeeds
```

3.1.5 Rebuilding a File System Image

When rebuilding the system, you can reuse the previously downloaded packages (about 450 MB) for the next build with the command:

```
make DL=~/openWrtFeeds TARGET=rpi_version
```

3.1.6 Programming the Raspberry Pi

After a successful make process, your image can be found at:

```
~/JN-AN-1222/Source/Host/openWrt/openWrt-
xx.yy/bin/brcm2708/openwrt-brcm2708-sdcard-vfat-ext4.img
```

where xx.yy represents the version of the OpenWrt system. Follow the instructions in the Raspberry Pi Installing Guide (3) to flash the SDCard image.

3.1.7 Updating Gateway Host Software

The complete source code of the Gateway Host software is provided within this Application Note package. NXP provides the source as an OpenWrt package, located in the directory Source/Host/openWrt/packages/NXP.

During system image build process, this package is copied into the openWrt build-root, e.g. in the location Source/Host/openWrt/openWrt-xx.yy/packages/NXP, where xx.yy represents the version of the OpenWrt system.

It can then be used as a starting point for further software developments. A complete description of the source code structure is available in Section 5, IoT Gateway Software.

After updating the source code, it needs to be recompiled. The commands to do this are:

```
cd ~/JN-AN-1222/Source/Host/openWrt/openWrt-xx.yy
make package/iot_gw/{clean,compile,install}
```

When the source has been built, the file system image can be updated with:

```
cd ~/JN-AN-1222/Source/Host/openWrt
make
```

Then proceed with flashing the SD Card.

4 Miscellaneous

4.1 Access to IoT Gateway System Console

For various reasons, such as controlling process execution, it might be useful to log into the Gateway. This can be done in two ways. In both cases, you will need access to a Linux command line interface running the BusyBox shell (4).

4.1.1 Local Console Access

The Raspberry Pi board has HDMI and USB ports. If you connect an HDMI monitor and a USB keyboard then you will have access to the system console.

4.1.2 Remote Console Access

You can have remote access to the Raspberry Pi board using an SSL connection. On a Windows PC, a tool like Tera Term (5), PuTTY (6) or any SSH (7) command line tool may help.

In your Linux development environment, you can use the ssh command line tool.

The credentials to connect are:

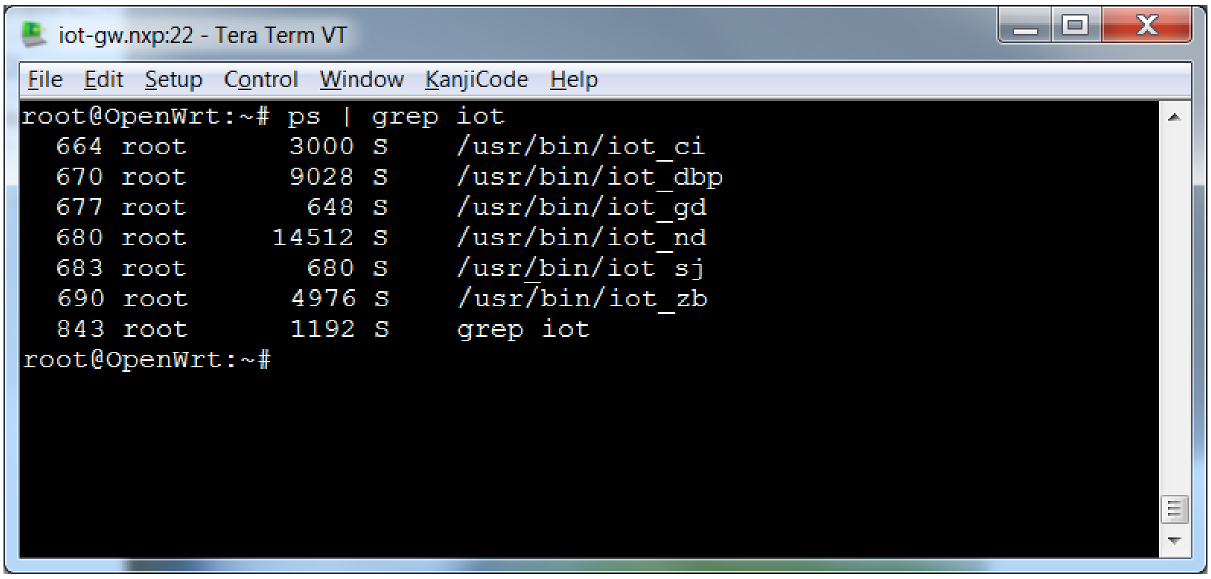
Login	root
Password	snap

4.2 IoT Processes Check

Open a new terminal (PuTTY) window to the IoT Gateway and run the following command:

```
ps | grep iot
```

Your output should contain all the IoT daemons, as shown in the screenshot below.


iot-gw.nxp:22 - Tera Term VT
File Edit Setup Control Window KanjiCode Help

```
root@OpenWrt:~# ps | grep iot
  664 root       3000 S    /usr/bin/iot_ci
  670 root       9028 S    /usr/bin/iot_dbp
  677 root        648 S    /usr/bin/iot_gd
  680 root      14512 S    /usr/bin/iot_nd
  683 root        680 S    /usr/bin/iot_sj
  690 root       4976 S    /usr/bin/iot_zb
  843 root       1192 S    grep iot
root@OpenWrt:~#
```

4.3 Changing the ZCB Image in a Running System

4.3.1 Re-flashing the ZCB USB Dongle

At the very first boot, the ZCB USB dongle is updated with a version of the ZCB firmware contained in the Gateway image. When the ZCB image needs to be updated at a later stage, be sure to first stop the iot_zb daemon that connects to the ZigBee USB dongle and then flash the previously downloaded image to "/tmp". Then start the IoT programs again:

```
/etc/init.d/iot_zb_initd stop

cp [new_firmware.bin]
     /usr/share/iot/ZigbeeNodeControlBridge_3v0_JN51xx.bin #
     xx={69,79}

/usr/bin/JennicModuleAutoProgram.sh

/etc/init.d/iot_zb_initd start
```

Note: The Control Bridge binary must have the specific name **ZigbeeNodeControlBridge_3v0_JN51xx.bin** and be located in **/usr/share/iot**.

Since the Flashing programming tool is a rather complex piece of software, it deserves some extra attention. The Flashing programming software contains several parts: code to control the bootloader of the JN51xx device, code to extend the bootloader functionality, the actual flashing code itself, and a shell script that puts all the parts together. The following sections detail how the update process is handled.

4.3.2 Controlling the Bootloader

The program that puts the JN51xx bootloader into program mode (or back) is called iot_jf.

It has the following options:

- iot_jf prog: put the bootloader into programming mode
- iot_jf normal: put the bootloader back into normal mode, which gives the UART back to the application.

Note: An awkward side-effect of this program is that it removes the /dev/ttyUSB0 device, so we need to repair this after the call (see Section 4.3.5, Shell Script).

4.3.3 Extending the Bootloader

After a new ZCB image has been flashed, and prior to starting it, the JN51xx EEPROM must be erased. The EEPROM-erase command is not available in the standard bootloader, so directly after flashing the new program, the bootloader functionality is extended with a piece of software called a Flash Programmer Extension. It is stored in directory /usr/share/iot/.

4.3.4 Flashing the JN51xx

The actual Flash programming program is based on the JN51xx Production Flash Programmer (JN-SW-4107). The flash program is called iot_jp and is called as follows:

```
iot_jp -I 38400 -P 1000000 -s /dev/ttyUSB0 -f <binary file> -v -V
2
```

The options are:

- `-I <baud>`: Initial baud rate of the bootloader, must be 38400
- `-P <baud>`: The programming baud rate, set to a million baud
- `-s <dev>`: The name of the serial port to which the JN51xx is connected
- `-f <binfile>`: The binary file that contains the new ZCB image
- `-v`: Verify after programming the binary file
- `-V <level>`: Verbosity level

4.3.5 Shell Script

The shell script that puts all the parts together is called JennicModuleAutoProgram.sh and performs the following steps:

1. Switch the bootloader to the programming state (iot_jf)
2. Re-install the ftdi_sio driver to get /dev/ttyUSB0 back
3. Call the Flash programmer (iot_jp)
4. Switch the bootloader back to the normal state (iot_jf)
5. Re-install the ftdi_sio to get /dev/ttyUSB0 back
6. Report the flashing result

This script is run at the very first boot of the Gateway. However, it can be executed at any time by accessing the Gateway System Console (see Section 4.1 Access to IoT Gateway System Console). The Control Bridge firmware can then be put in the Gateway file system using, for example, the scp command (Linux) or pscp command (Windows). Section 4.3.1 provides additional information on how to update the ZCB firmware.

When called manually from the command line, the programming and verification progress can be executed in sequence.

4.4 Reverting to the Default ZCB Image

At first boot, the ZigBee Control Bridge is flashed with the relevant binary file from the folder /usr/share/iot. Once successfully flashed, the following file is created to avoid reflashing the ZCB at each boot:

```
/usr/share/iot/.zcb_flashed
```

In the case in which the Control Bridge has been flashed with an external tool, it is possible to revert to the default ZCB firmware by deleting the file `/usr/share/iot/.zcb_flashed` and rebooting the Gateway.

4.5 Changing the ZigBee Protocol Version

By default, the software package is configured to use ZigBee protocol version 3.0. It is possible to switch back to ZigBee 2.0 at Host image build-time or at run-time.

4.5.1 Changing at Host Image Build-time

To change the ZigBee protocol version at image build-time, use the command set below (from the buildroot directory), after having built the image at least once:

```
make menuconfig
```

Then go to the menu:

```
NXP ->
    Internet of Thing - Gateway software ->
           Select Zigbee protocol version for Control Bridge
```

and select appropriate ZigBee protocol version.

4.5.2 Changing at Run-time

To change ZigBee protocol version at run-time, use the command set below (from the target's Linux command line interface):

```
echo _2v0 > /usr/share/iot/ZigbeeProtocolVersion.txt
```

_2v0 can be replaced by _3v0 as well.

Then apply the instructions from Section 4.4.

5 IoT Gateway Software

5.1 Source Code File Structure

All the source files for the IoT Gateway are located under directory

```
Source/Host/openWrt/packages/NXP/iot_gw/src
```

The table below depicts how files are organized in the provided zip archive. Path descriptions start from the directory

```
src
```

Path		Description
daemons/		Contains the source code related to daemons
	ControlInterface/	Control Interface daemon
	dbp/	Database manager
	nfcRpiAlt/	NFC daemon
	SecureJoiner/	Secure joiner daemon
	ZCB/	ZigBee Control Bridge daemon
IotCommon/		The IotCommon directory contains all files related to common features of daemons and CGI scripts.
jnFlasher/		Contains the code to build JN5169 flasher for the Linux environment.
www/		Contains all the files related to the web interface.
	cgi-bin/	CGI scripts are located here
	css/	Cascading Style Sheets, for web pages.
	img/	Image repository, for web pages.
	js/	Java Script repository
Zcb/		Contains the ZigBee Control Bridge images. These binaries are built from another development environment and has to be hand-copied into this location each time a new version is available. The package contains binaries for JN5168, JN5169 and JN5179, and for the ZigBee protocol version 2.0 and 3.0.

5.2 Building IoT Software for Another Target

The source code provided with the Application Note is provided as an OpenWrt software package. However, the following sections explain how to build all the pieces of software for an embedded Linux systems other than OpenWrt.

5.2.1 Cross-compile Environment Variables

In the directory Source/Host/openWrt/packages/NXP/iot_gw/src there is a makefile (Makefile) which can be used to cross-compile the source code to almost any Linux platform. This makefile relies on the following environment variables:

Environment Variables	Content Description
TARGET_MACHINE	Used for file and code selection. Must be equal to RASPBERRYPI
TARGET_OS	Used for file and code selection. Must be equal to OPENWRT
CFLAGS	Contains target specific flags. Must end with : -DTARGET_RASPBERRYPI

	-DTARGET_OPENWRT
CC	Path to C compiler
CXX	Path to C++ compiler
AR	Path to archiver
LD	Path to linker
ZCB_VERSION	Contains one the values below: _3v0 : for ZigBee version 3.0 _2v0 : for ZigBee version 2.0

The table above shows the environment configuration that needs to be set to select the correct cross-compiler and linker for the embedded Linux target.

File Source/Host/openWrt/packages/NXP/iot_gw/Makefile provides an example of how to set up these environment settings.

Note: For IoT Gateway development, it is important to set the environment according to your environment.

5.2.2 Top-level makefile

In the same directory, Source/Host/openWrt/packages/NXP/iot_gw/src, the top-level makefile also supports different targets. When it is executed, all sub-directories will be built recursively. The supported targets are detailed in the table below.

Targets	Content
build	Compiles images
all	A combination of clean and build
clean	Remove all .o files, executable binaries and generated scripts.

5.2.3 Use of IoT Gateway Target Directories

The following target directories are used by the IoT Gateway software:

Directory	Content
/usr/share/iot	Contains the IoT database Also contains the software version text and JSON files for the test programs
/usr/bin	IoT programs, daemons and scripts
/etc/init.d	Init scripts to start or stop the IoT daemons
/tmp	For log files and message queues
/www	Modified index.html
/www/css	IoT CSS files
/www/js	IoT JavaScript files
/www/img	IoT web page images

Note: Calling make clean first before make build will ensure that everything is built again from scratch.

5.2.4 IoT Specific Settings

A number of directories have to be created, using the commands below:

```
mkdir /www/img
mkdir /www/css
mkdir /www/js
mkdir /www/cgi-bin
mkdir /usr/share/iot
```

The right document paths must be set in the Apache configuration file. To do this:

1. Edit the Apache configuration file using the command below:
   ```
   vi /etc/apache/httpd.conf
   ```
2. Change some of the paths as specified below:
   ```
   /usr/share/htdocs -> /www
   /usr/share/cgi-bin -> /www/cgi-bin
   ```
3. Save the file.
4. Update access rights on the directory /tmp so that for software updates can be done:
   ```
   chmod o+w /tmp
   ```
5. Start the Apache web server:
   ```
   apachectl start
   ```
6. Make sure that Apache also starts after a reboot, by editing the file /etc/rc.local:
   ```
   vi /etc/rc.local
   ```
   and adding the following line:
   ```
   apachectl start
   ```

5.2.5 IoT Gateway Initialization

When all items are copied to the target, the initialization scripts (required once after each upload) must be run to set everything correctly for start-up. The initialization script (see the appendix "Initialization Script After Upload") does the following:

- Creates some IoT directories if they do not already exist
- Sets the permissions correctly for the uploaded images
- Sets the write permissions on /tmp for the update mechanism
- Creates automatic start-up scripts for new IoT programs/daemons
- Upgrades the new Control Bridge image

This script can be run via a SSH terminal window on the IoT Gateway or typed locally (see Section 4.1 Access to IoT Gateway System Console).

6 Release Details

6.1 Compatibility

Product Type	Part Number	Build
Version 2.2		
Evaluation Kit	JN516x-EK004	-
Evaluation Kit	JN517x-DK005	-

6.2 New Features

Feature	Description
Version 2.2	
None	
Version 2.1	
JN5179 ZCB	Added support for JN5179 ZigBee Control Bridge for JN517x-DK005
Version 2.0	
ZigBee 3.0	Added support for ZigBee protocol version 3.0.

6.3 Known Issues

ID	Severity	Description
Versions 2.1 and 2.2		
lpap799	Minor	Smart plug power consumption not correct (add Configure Reporting)
lpap811	Minor	Unexpected jumps in web interface at on/off toggling for lamps

6.4 Bug Fixes

ID	Description
Versions 2.1 and 2.2	
None	

Appendices

Initialization Script After Upload

```
#!/bin/sh
# -------------------------------------------------------------------
# Author: nlv10677
# Copyright: NXP B.V. 2015. All rights reserved
# -------------------------------------------------------------------
# ----------------------------------------------------------------
# Create some directories if they not already exists
# ----------------------------------------------------------------
echo "+++++++++ Init start +++++++" >> /tmp/su.log
echo "Init 1: Make IoT directories" >> /tmp/su.log
if [ ! -d /usr/share/iot ]; then
mkdir /usr/share/iot
fi
if [ ! -d /www/img ]; then
mkdir /www/img
fi
if [ ! -d /www/js ]; then
mkdir /www/js
fi
if [ ! -d /www/css ]; then
mkdir /www/css
fi
if [ ! -d /www/cgi-bin ]; then
mkdir /www/cgi-bin
fi
# ----------------------------------------------------------------
# Set the persmissions right of our uploaded images
# ----------------------------------------------------------------
echo "Init 2: Set permissions right" >> /tmp/su.log
chmod +x /etc/init.d/iot_*
chmod +x /usr/bin/iot_*
chmod +x /usr/bin/killbyname
chmod +x /www/cgi-bin/iot_*
# ----------------------------------------------------------------
# Set the write persmissions on /tmp for the update mechanism
# ----------------------------------------------------------------
echo "Init 3: Make /tmp writable for Apache" >> /tmp/su.log
chmod o+w /tmp
# ----------------------------------------------------------------
# Create automatic startup scripts for our new IoT programs/daemons
# The test is there to allow multiple calls of this script
# ----------------------------------------------------------------
echo "Init 4: Install startup scripts" >> /tmp/su.log
if [ ! -f /etc/rc.d/S99iot_zb_initd ];
then
/etc/init.d/iot_zb_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_dp_initd ];
then
/etc/init.d/iot_dp_initd enable
```

```
fi
if [ ! -f /etc/rc.d/S99iot_sj_initd ];
then
/etc/init.d/iot_sj_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_ci_initd ];
then
/etc/init.d/iot_ci_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_nd_initd ];
then
/etc/init.d/iot_nd_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_gd_initd ];
then
/etc/init.d/iot_gd_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_dbp_initd ];
then
/etc/init.d/iot_dbp_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_su_initd ];
then
/etc/init.d/iot_su_initd enable
fi
```

Init Script

```
#!/bin/sh
# ------------------------------------------------------------------
# Author: nlv10677
# Copyright: NXP B.V. 2015. All rights reserved
# ------------------------------------------------------------------
# ---------------------------------------------------------------
# Create some directories if they not already exists
# ---------------------------------------------------------------
echo "+++++++++ Init start +++++++" >> /tmp/su.log
echo "Init 1: Make IoT directories" >> /tmp/su.log
if [ ! -d /usr/share/iot ]; then
mkdir /usr/share/iot
fi
if [ ! -d /www/img ]; then
mkdir /www/img
fi
if [ ! -d /www/js ]; then
mkdir /www/js
fi
if [ ! -d /www/css ]; then
mkdir /www/css
fi
if [ ! -d /www/cgi-bin ]; then
mkdir /www/cgi-bin
fi
# ---------------------------------------------------------------
# Set the persmissions right of our uploaded images
# ---------------------------------------------------------------
echo "Init 2: Set permissions right" >> /tmp/su.log
chmod +x /etc/init.d/iot_*
chmod +x /usr/bin/iot_*
chmod +x /usr/bin/killbyname
chmod +x /www/cgi-bin/iot_*
# ---------------------------------------------------------------
# Set the write persmissions on /tmp for the update mechanism
# ---------------------------------------------------------------
echo "Init 3: Make /tmp writable for Apache" >> /tmp/su.log
chmod o+w /tmp
# ---------------------------------------------------------------
# Create automatic startup scripts for our new IoT programs/daemons
# The test is there to allow multiple calls of this script
# ---------------------------------------------------------------
echo "Init 4: Install startup scripts" >> /tmp/su.log
if [ ! -f /etc/rc.d/S99iot_zb_initd ];
then
/etc/init.d/iot_zb_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_dp_initd ];
then
/etc/init.d/iot_dp_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_sj_initd ];
then
/etc/init.d/iot_sj_initd enable
```

```
fi
if [ ! -f /etc/rc.d/S99iot_ci_initd ];
then
/etc/init.d/iot_ci_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_nd_initd ];
then
/etc/init.d/iot_nd_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_gd_initd ];
then
/etc/init.d/iot_gd_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_dbp_initd ];
then
/etc/init.d/iot_dbp_initd enable
fi
if [ ! -f /etc/rc.d/S99iot_su_initd ];
then
/etc/init.d/iot_su_initd enable
fi
# ---------------------------------------------------------------
# Upgrade the new Coordinator image
# ---------------------------------------------------------------
echo "Init 5: Check for Coordinator image" >> /tmp/su.log
if [ -f /tmp/ZigbeeNodeControlBridge_JN5169.bin ];
then
echo "Init 6: Upgrade Coordinator image" >> /tmp/su.log
JennicModuleProgram.sh /tmp/ZigbeeNodeControlBridge_JN5169.bin
# Remove it to save space on /tmp
rm /tmp/ZigbeeNodeControlBridge_JN5169.bin
fi
echo "+++++++++ Init 7: end +++++++" >> /tmp/su.log
```

References

1. OpenWrt : buildroot exigence, install procedure on linux. OpenWrt.org. [Online] http://wiki.openwrt.org/doc/howto/buildroot.exigence#install_procedure_on_linux.
2. OpenWrt : Build with VM. OpenWrt.org. [Online] http://wiki.openwrt.org/doc/howto/buildvm.
3. installing-images. Raspberry Pi. [Online] https://www.raspberrypi.org/documentation/installation/installing-images.
4. BusyBox. [Online] http://www.busybox.net/.
5. Teraterm. [Online] http://ttssh2.osdn.jp/.
6. PuTTY. [Online] http://www.putty.org/.
7. Secure Shell. Wikipedia. [Online] https://en.wikipedia.org/wiki/Secure_Shell.
8. NFC commissioning.
9. JN-AN-1222 Zigbee IoT Gateway with NFC - User guide.
10. JN-AN-1223 Application Note - ZigBee IoT Control Bridge.
11. JN-AN-1216 Application Note - ZigBee 3.0 IoT Control Bridge.

Revision History

Version	Release/Tag	Notes
1.0		First release
1.1		Added support for Raspberry Pi 2, fixed typo errors
2.0		Added support for ZigBee 3.0
2.1		Added support for JN517x-DK005 (JN5179 Control Bridge)
2.2		Editorial updates made
2.3		Mark Raspberry Pi Model B+ as obsolete (unsupported)
2.4		Add reference to JN-AN-1216 (ZCB ZigBee 3.0)
2.5	v2006	Update ZCB ZigBee 3.0 binary files from JN-AN-1216 (1004)
2.6	v2007	Support of Installation Code NFC commissioning

Important Notice

NXP Semiconductors

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

www.nxp.com