NXP Semiconductors
User's Guide

Document Number: BLEIP6UG
Rev. 0, 09/2016

Bluetooth® Low Energy Transport for IPv6 Datagrams

User's Guide

1. Introduction

This document describes how devices running the Kinetis Bluetooth® Low Energy stack can create or join an IPv6 network and transport IPv6 datagrams.

The document first presents an overview of transporting IPv6 over Bluetooth as presented by RFC 9776. It details the implemented protocol stack, supported topologies and IPv6 specifics or this transport.

Chapter 3 presents the structure of the example applications, to make the user more familiar with the IPv6 specific modules used and also details some of the main APIs.

Chapter 4 depicts how to configure and initialize the nodes to fulfill the desired functionality.

The last chapter describes step by step how the user can run the example to showcase features like ICMPv6, UDP, CoAP or Sockets.

Contents

2. IPv6 over Bluetooth Overview

The Internet of Things assumes connectivity for all nodes, so the ideal protocol for connecting billion devices is IPv6. The transport of IPv6 datagrams on top of a Bluetooth Host Stack is defined by the IETF and depicted in RFC 7668.

2.1. Protocol Stack

The IPv6 over Bluetooth solution protocol stack is presented in Figure 1.

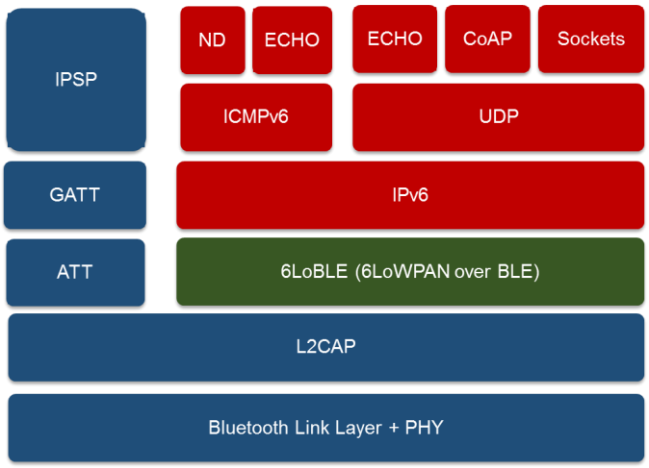

Figure 1. IPv6 over Bluetooth Protocol Stack

The Bluetooth stack, depicted in blue, must implement a GATT-based profile called Internet Protocol Support Profile (IPSP) that specifies the discovery and connection of other supporting nodes. Furthermore, the transport of IPv6 datagrams is done on L2CAP Credit Based channels. Therefore, a Bluetooth Host Stack version of 4.1 ar above is required.

The adaptation of IPv6 datagrams over the Bluetooth interface is done by using 6LoWPAN techniques described in RFC 7668. The 6LoWPAN over BLE layer, presented in green is also responsible for IP or UDP header compression of IPv6 packets, for optimizations of neighbor discovery or multicast, as well as for stateless address autoconfiguration (SLAAC). This layer is named SixLoBle inside the Network IP stack.

The IPv6 layers are presented in red and comprise the Network IP stack. It supports Internet Control Message Protocol (ICMPv6) and User Datagram Protocol (UDP) as a transport layer. The SDK offers support for other Application layers like Sockets, Constrained Application Protocol (CoAP), Echo. Neighbor Discovery Protocol (ND) is also implemented and is used for prefix dissemination and address registration.

2.2. Topologies

IPv6 over Bluetooth supports only a hub-and-spokes topology, as depicted in Figure 2. Therefore, two types of devices exist:

- 6LBR – 6LoWPAN Border Router which is an IPSP Router and a GAP Central.
- 6LN – 6LoWPAN Node which is an IPSP Node and a GAP Peripheral.

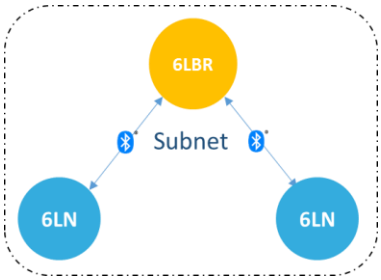

Figure 2. Isolated Bluetooth IPv6 Network

All 6LN devices and the Bluetooth interface of the 6LBR are in the same IPv6 subnet. Optionally, the 6LBR may have an additional interface over Ethernet to communicate with the Internet, as presented in Figure 3.

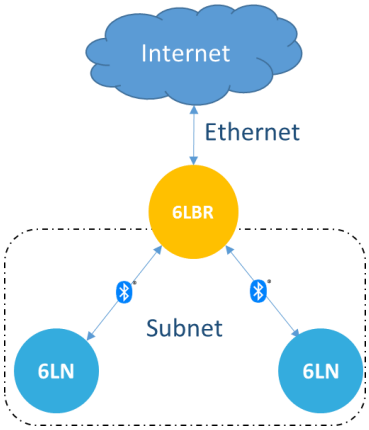

Figure 3. Bluetooth IPv6 Network with Internet Connection

2.3. Node Configuration and Registration

Once a Bluetooth Link is established between the GAP Central and the GAP Peripheral, the 6LBR tries to open an L2CAP Credit Based Channel is set up. After the channel is created, the devices opens the 6LoWPAN interfaces and assignes an IPv6 link-local address based on the Bluetooth link layer addres. This procedure is called Stateless address auto-configuration (SLAAC) and is done in two steps:

1. A 64-bit Interface Identifier (IID) is formed from the 48-bit Bluetooth device address by inserting 0xFFFE in the middle of the 48 bits. According to RFC2373, when creating an IID, the universal/local bit (“u” bit) is set to one (1) when it indicates a global scope (for Bluetooth public addresses) and to zero (0) to indicate local scope (when using Bluetooth random addresses).
2. The IPv6 link-layer address is created by prepending the FE80::/64 prefix.

For example, let's take a device with a Bluetooth public address of:

00	04	9F	00	00	0B

Adding the 0xFFFE, and setting the "u" bit to 1, yields:

02	04	9F	FF	FE	00	00	0B

After prepending the predefined prefix, the IPv6 address is created:

FE	80	00	00	00	00	00	00	02	04	9F	FF	FE	00	00	0B

After the 6LN has a link-local IPv6 address, it will start router discovery by sending an ICMPv6 Router Solicitation message (RS). The router discovered will always be the 6LBR which will respond with an ICMPv6 Router Advertisement (RA)containing the following information, depending on the configuration:

- Prefix Information Option (PIO): Contains a Global or Unique Local Prefix configured on the router.
- 6CO: 6LoWPAN Context Option: It carries prefix information for header compression
- On-link flag ("L") is always set to zero to signal the 6LN to always send packets to the 6LBR.

After receiving the RA, the 6LN configures a global IPv6 address by using the same SLAAC procedure but with the received global prefix, instead of the link-local one. After creating the address, it will register it at the router by sending an ICMPv6 Neighbor Solicitation (NS) with an address registration option (ARO) containing the 64-bit Extended Unique Identifier (EUI-64). The 6LBR will respond with an ICMPv6 Neighbor Advertisement with the ARO field containing a status field that describes the registration result.

The procedure is detailed in Figure 3:

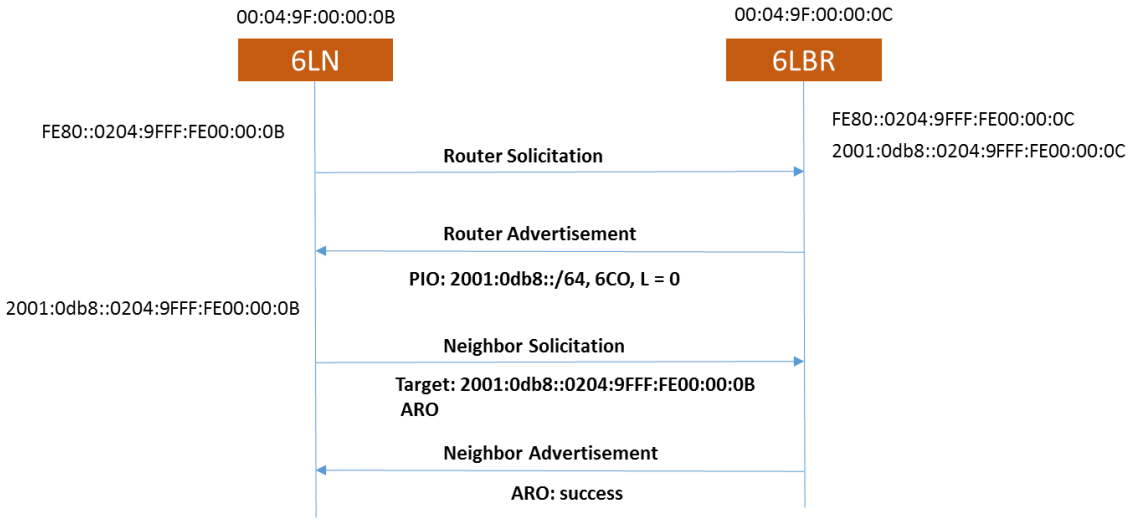

Figure 4. 6LN Configuration and Registration

2.4. IPv6 Datagram Transport Details

Due to the nature of the Bluetooth Link Layer and topology, ther are some specific rules that apply to this IPv6 adaptation.

The 6LN node will always send the IPv6 packet to its 6LBR peer.

Ttraffic between the nodes is always unicast. Therefore multicast from a 6LN is transformed in a unicast to the 6LBR and multicast form 6LBR is transformed in multiple unicast frames sent to connected 6LNs that have registered for multicast.

Header compression is realized by following the guidelines from RFC 6282, which specifies the compression format for IPv6 datagrams on top of IEEE 802.15.4.The efficiency of the compression increases when:

- An RA with 6LoWPAN Context Option (**6CO**) matching each **PIO** is received, or
- Address registration with (**ARO**) is done.

Elision of source IPv6 address, destination IPv6 address or both translates in saving:

- Full IP Address (on link-local or last registered) – 128 bits,
- Prefix when compression context available – 64 bits,
- IID – 48 bits.

3. Applications Structure and APIs

The application follows the main structure of the other Bluetooth applications in the SDK, as described in the Bluetooth Low Energy Application Developer's Guide document. The additions for these examples include the *nwk_ip* folders one in the root of the application (containing the Network IP library interface and media interface), one in the *app/common* folder (containing the Network IP Manager andand two configuration files: *app_stack_config.h* and *nwk_ip_config.h*, as presented in Figure 5:

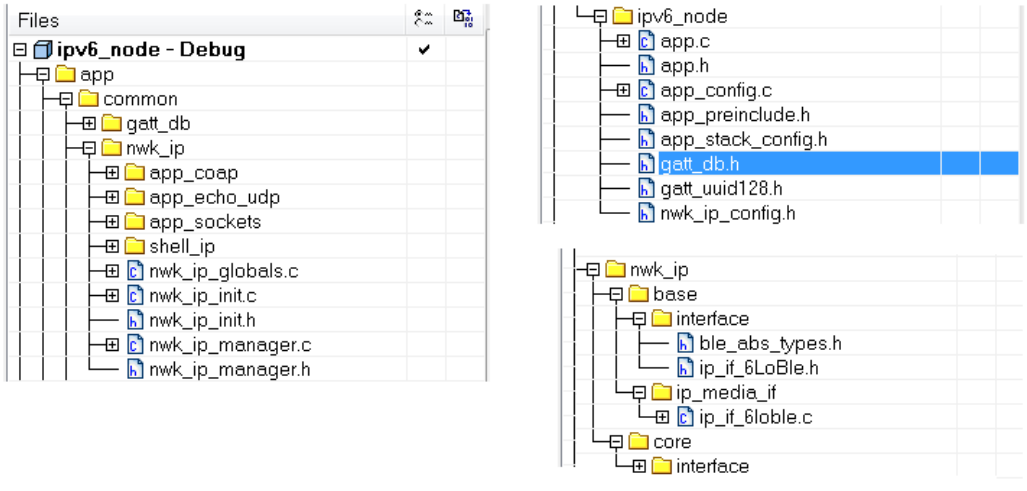

Figure 5. Application Structure of an IPv6 Node example

3.1. Network IP Library and Media Interface

The *nwk_ip* folder contains the public interface for the core modules inside the library like ND or SixLoBLE and also the source file and headers for the media interface: *ip_if_6loble.c*.The module contains code for opening, closing and sending data on the media interface.

It is recommended not to modify these files.

3.2. Network IP Manager and Protocol Wrappers APIs

The nwk_ip folder inside the *app/common* folder contains the source and header files for the following modules:

- Network IP Manager,
- Network IP Task Initialization,
- LED example application running CoAP,
- ECHO UDP example,
- Sockets example
- Network IP globals.
- Shell menus.

It is recommended not to modify these files.

3.2.1. Network IP Manager API

The Network IP Manager is the module that helps the application initializes and configures the IPv6 layers inside the IP stack library or their respective wrappers. It consists of 3 major APIs.

```
nwkStatus_t NwkIpManager_Init(void);
```

This API initializes common modules for both IPv6 Router and Node devices. It initializes the SixLoBLE module and instance. It then registers the Bluetooth callbacks needed by the module to send data on the bearer and links the media interface to the IP layer. Lastly it initializes the application wrappers for CoAP, Sockets and UDP Echo. It return a status of `gNwkStatusSuccess_c` or an error.

```
nwkStatus_t NwkIpManager_StartNode(void);
```

This API configures modules for IPv6 Node devices. It registers the unicast forward procedure, opens the ND channel and sets the registration lifetime of the node. It return a status of `gNwkStatusSuccess_c` or an error.

```
nwkStatus_t NwkIpManager_StartRouter(void);
```

This API configures modules for IPv6 Router devices. It registers the unicast forward procedure, opens the ND channel, adds the initial global prefix to the interface and in the context compression table. The node also configures as a default router. It return a status of `gNwkStatusSuccess_c` or an error.

3.2.2. Network IP RTOS Prerequisites

The *nwk_ip_init* module contains the RTOS initialization part of the IP Task. To initialize the task, the application must call the following API, at startup (usually this is done in *BleApp_Init*):

```
osaStatus_t NwkIpManager_TaskInit(void)
```

The task size and priority shall be configured in nwk_ip_init.h:

```
#ifndef gIpStack_TaskStackSize_c
#define gIpStack_TaskStackSize_c 3000
#endif

#ifndef gIpStack_TaskPriority_c
#define gIpStack_TaskPriority_c  5
#endif
```

The IP Task is signaled by the NWKU_GENERIC_MSG_EVENT event flag and uses three queues: one for SixLoBle layer , one for the IP layer, and one for the Network IP wrapper layer (including ECHO ICMPv6 or UDP, Sockets or CoAP).

3.2.3. CoAP LED Example

The Constrained Application Protocol (CoAP) is a web transfer protocol used on resource-constrained nodes and networks.This application layer protocol is one of the most used protocols for Machine to Machine (M2M) communication in home automation deployments, part of the Internet of Things ecosystem.

CoAP is based on a Representational state transfer (REST) model similar to the client-server model of HTTP. It uses the same HTTP verbs for retrieving and sending data, such as GET, POST, PUT, DELETE. It runs over an UDP transport layer. One of the main features that make this protocol the best pick for a wireless sensor network are:

- Low overhead and parsing complexity
- Support for discovery of resouces (usually multicast)
- URI support
- Support for selection of data models(XML, JSON, and others)

CoAP also defines four types of messages: Confirmable, Non-confirmable, Acknowledgement and Reset.

The SDK provides a high level CoAP over UDP module, with an demo functionality of modifiying the configuration the the RGB LED on the boards. The resource is identified by the */led* URI. The peer devices can POST messages to this resource to change the state of the LED: on, off, toggle, flash, rgb.

To initialize the module, the following API must be called. The function inityializes a callback configuration structure and registers services to the CoAP modules by creating the CoAP instance. In our example, this module is called by the Network IP Manager.

```
void APP_InitCoap(void)
{
   coapRegCbParams_t cbParams[] =  {{APP_CoapLedCb,  (coapUriPath_t*)&gAPP_LED_URI_PATH}};
   /* Register Services in COAP */
   coapStartUnsecParams_t coapParams = {COAP_DEFAULT_PORT, AF_INET6};
   mCoapInstId = COAP_CreateInstance(NULL, &coapParams, gIpIfSlp0_c,
       (coapRegCbParams_t*)cbParams, NumberOfElements(cbParams));
}
```

The callback function processes the POST command and sends an acknowledgement back:

```
static void APP_CoapLedCb
(
    coapSessionStatus_t sessionStatus,
    void *pData,
    coapSession_t *pSession,
    uint32_t dataLen
)
{
    /* Process the command only if it is a POST method */
    if ((pData) && (sessionStatus == gCoapSuccess_c) && (pSession->code == gCoapPOST_c))
    {
        APP_ProcessLedCmd(pData, dataLen);
    }

    /* Send the reply if the status is Success or Duplicate */
    if ((gCoapFailure_c != sessionStatus) && (gCoapConfirmable_c == pSession->msgType))
    {
        /* Send CoAP ACK */
        COAP_Send(pSession, gCoapMsgTypeAckSuccessChanged_c, NULL, 0);
    }
}
```

The functionality is exercised in the Shell Network IP module. For more information, please read the *Kinetis Thread Stack Application Development Guide* and the *Kinetis Thread Stack API Reference Manual*.

3.2.4. Socket Data API

The socket API allows creating generic BSD/POSIX-style IP sockets over the IPv6 network and to exterior network destinations.
The following code block shows how to create a UDP socket and bind it to any local IPv6 address and port:

```
static void APP_InitSocketServer(void)
{
    /* Socket storage information used for RX */
    sockaddrIn6_t mSsRx = {{{0}}};

    if(mSockfd != -1)
```

```
    {
        /* socket already initialised */
        return;
    }

    /* Create a socket for global IP address */
    /* Set local information */

    mSsRx.sin6_family = AF_INET6;
    mSsRx.sin6_port = UDP_PORT;
    IP_AddrCopy(&mSsRx.sin6_addr, &in6addr_any);

    /* Create socket */
    mSockfd = socket(mSsRx.sin6_family, SOCK_DGRAM, IPPROTO_UDP);

    /* Bind socket to local information */
    bind(mSockfd, (sockaddrStorage_t*)&mSsRx, sizeof(sockaddrStorage_t));

    /* Initializes the Session task*/
    Session_Init();

    Session_RegisterCb(mSockfd, APP_SocketClientRxCallback, pmAppSockThreadMsgQueue);
}
```

This function is called to initialize the wrapper module. For sending a generic Socket application request the following API is called:

```
bool_t App_SocketSendAsync(appSockCmdParams_t* pAppSockCmdParams);
```

The functionality is exercised in the Shell Network IP module. For more information, please read the *Kinetis Thread Stack Application Development Guide* and the *Kinetis Thread Stack API Reference Manual*.

3.2.5. Echo Data UDP

The module adds Echo capabilities for a default socket that is created at initialization (ECHO_SERVER_PORT). The functionality is exercised in the Shell Network IP module.

3.2.6. Shell Menus

The module contains the shell functionality of the example applications. It gives the user capability to send data and also feedback on received information from peer devices. For more details, enter the *help* keyword:

```
BLE IPv6 Node>help
help                print command description/usage
version             print version of all the registered modules
history             print history

ifconfig            IP Stack interfaces configuration
mcastgroup          Multicast groups management
ping                IP Stack ping tool
```

```
coap                Send CoAP message
socket              IP Stack BSD Sockets commands
echoudp             Echo udp client
```

3.2.7. Network IP Globals

This module contains all structures used by the Network IP Stack and dimensioned or initialized by the application. For example: UDP connections table, CoAP instance, prefix tables and others

3.3. Configuration files

The user can configure the Network IP stack and Manager by using the two configuration files: *app_stack_config.h* and *nwk_ip_config.h*.

app_stack_config.h is used to configure variables declared in the globals section described earlier, like the number of sockets , UDP connections or compression contexts, as depicted below:

```
/*! The maximum number of UDP connections that can be opened at one time. MUST not be
greater than
 * BSDS_MAX_SOCKETS */
#ifndef MAX_UDP_CONNECTIONS
    #define MAX_UDP_CONNECTIONS                 10
#endif
```

nwk_ip_config.h is used to configure variables used by the Network IP Manager, like the ND PIB or the initial global prefix. Below you can find the configuration of an IPv6 Router:

```
#define GLOBAL_PREFIX_INIT \
        {0x20, 0x01, 0x0d, 0xb8, 0x00, 0x00, 0x00, 0x00, \
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}

#define GLOBAL_PREFIX_INIT_LEN  64
```

The user should add in these header files any extra configurations.

4. IPv6 Network Setup

4.1. L2CAP and GATT Prerequisites

The transport of IPv6 datagrams is made over L2CAP Credit Based channels. To setup this feature, the user will first register the IPSP Protocol/Service Multiplexer(PSM) after the Bluetooth stack is initialized:

```
static void BleApp_Config()
{
...
    /* Register IPSP PSM */
    L2ca_RegisterLePsm(gL2ca_Le_Psm_Ipsp_c, gL2capMtu_d);
}
```

The link between the L2CAP layer and the 6LoBle layer is done by registering the data and control callbacks. Also, *bleAbsReq* contains function pointers that are called by the 6LoBle layer to send data over the BLE medium or query Bluetooth addresses or maximum SDU.

```
bleAbsRequests_t* BleApp_RegisterBleCallbacks(instanceId_t instanceId)
{
    mSixLoBleInstanceId = instanceId;
    L2ca_RegisterLeCbCallbacks(BleApp_IpspDataCallback, BleApp_IpspControlCallback);
    return (void *)&bleAbsReq;
}
```

The data callback creates a data indication message that is sent to the 6LoBle layer for processing:

```
static void BleApp_IpspDataCallback(deviceId_t deviceId, uint16_t channelId, uint8_t*
pPacket, uint16_t packetLength)
{
    bleDataInd_t* pBleDataInd = NULL;

    pBleDataInd = MEM_BufferAlloc(sizeof(bleDataInd_t));

    /* Create the BLE Data indication */
    if (pBleDataInd != NULL)
    {
        pBleDataInd->pSdu = MEM_BufferAlloc(packetLength);

        if (pBleDataInd->pSdu)
        {
            pBleDataInd->instanceId = mSixLoBleInstanceId;
            BleApp_GetBleAddress(pBleDataInd->dstAddr);
            BleApp_GetPeerBleAddress(deviceId, pBleDataInd->srcAddr);
            pBleDataInd->sduLength = packetLength;
            FLib_MemCpy(pBleDataInd->pSdu, pPacket, packetLength);

            /* Send message to 6lo */
            SixLoBle_DataIndCB(pBleDataInd);
        }
        else
        {
            MEM_BufferFree(pBleDataInd);
        }

    }
```

The GATT database of the IPv6 Node will include the Internet Protocol Support Service (IPSS):

```
PRIMARY_SERVICE(service_ipss, gBleSig_IpsService_d)
```

After it established the GAP connection, the IPv6 Router will first discover the IPSS Service:

```
uint16_t uuid = gBleSig_IpsService_d;
/* Start Service Discovery */
BleServDisc_FindService(peerDeviceId, gBleUuidType16_c, (bleUuid_t*) &uuid);
```

If the IPv6 Router discovers the service on the peer, it will then initiate an L2CAP connection on the IPSP PSM:

```
Ipsp_Connect(&mIpspRouterConfig, peerDeviceId);
```

4.2. Starting as an IPv6 Router

As soon as the Bluetooth Stack has been configured, the Network IP Manager module is initialized. This initializes the SixLoBle media interface and registers it to the IP layer. It also registers the current Bluetooth Link Layer to the 6lo interface. Furthermore it initializes the UDP Echo, Sockets and CoAP wrappers.

After the initialization, node is configured to start as an IPv6 Router. This starts and configures ND on the opened interface.

```
static void BleApp_Config()
{
...

    /* Initialize Network IP Manager module */
    NwkIpManager_Init();

    /* Start as an IPv6 Router */
    NwkIpManager_StartRouter();
}
```

4.3. Starting as an IPv6 Node

As soon as the Bluetooth Stack has been configured, the Network IP Manager module is initialized:

```
static void BleApp_Config()
{
...

    /* Initialize Network IP Manager module */
    NwkIpManager_Init();
}
```

After the IPSP L2CAP connection is established, the device si configured to start as an IPv6 Node:

```
case gL2ca_LePsmConnectionComplete_c:
{
   ...
   NwkIpManager_StartNode();
}
```

5. Running the IPv6 over Bluetooth Example

This section presents the user interactions, and testing methods for the IPV6 Router and Node applications.

5.1. General considerations

The IPV6 Router behaves as a GAP central node. It enters GAP Limited Discovery Procedure and searches for IPV6 nodes to connect with. When the IPV6 Router connects it initiates an LE Credit Based Connection request.

The IPV6 Node behaves as a GAP peripheral node. It enters GAP General Discoverable Mode and waits for a GAP central node to connect. When the IPv6 Router connects, it initiates an LE Credit Based Connection request.

When the credit based connection is set, the user can send data through the credit based channel from each device. When the device knows that the peer is out of credit, it automatically requests new credit to the application. The application then sends new credits to the peer. The examples use pairing with bonding by default. When connected with the IPv6 Node application, the IPv6 Router sends the 999999 passcode to the host stack by default.

The setup requires a minumim of two FRDM-KW41Z platforms, one for the IPv6 Node and one for the IPv6 Router. Optionally, a thrird board could be used for a second IPv6 Node connected to the IPv6 Router.

Open serial port terminals and connect them to the boards, in the same manner as described in Bluetooth Low Energy Demo Applications User's Guide. The start screen is displayed after the board is reset. At first, all LEDs are flashing on both devices.

5.2. User interface

After flashing the boards, the devices are in idle mode (all LEDs flashing).

To start scanning on the IPv6 Router, press the SW4 button. When in GAP Limited Discovery Procedure, LED3 is flashing. To start advertising on the IPv6 Node, press SW4. When in GAP Discoverable Mode LED3 is flashing.

When the nodes are connected, LED3 turns solid. To disconnect the nodes, hold the SW4 button pressed for 2-3 seconds on the IPv6 Node. The IPv6 Node re-enters GAP Discoverable Mode.

5.3. Creating the network

Press the SW4 button on the router board to start scanning for devices. Do the same on the node board to make it enter GAP Discoverable Mode. The host connects with the board after it sees it advertise the IPSP service and connects to it. A solid LED3 indicates a successful connection between the two devices. This is also visible in the console, as shown below.

The IPv6 Node will show it is now and has established a credit based channel.

```
Advertising...
Connected!


LE Credit Based Connection Complete!
   Device Id: 0
   Channel Id: 64
   Peer MTU: 1280
   Peer MPS: 251
   Credits: 50
```

As shown on the console, the IPv6 Router scans the IPv6 Node, connects to it, discovers the IPSS, initiates LE Credit Based Connection and establishes one.

```
Scanning...
```

```
Found device: IPv6_NODE 0x00049F00000B
Connecting...
Connected! Device Id: 0
Discovering IPSS...
Initiating LE Credit Based Connection request...
LE Credit Based Connection Complete!
   Device Id: 0
   Channel Id: 64
   Peer MTU: 1280
   Peer MPS: 251
   Credits: 50
```

5.4. Checking interfaces

To verify the interfaces and the addresses assigned, enter *ifconfig* or *ifconfig <interface>* as shown below.

All devices report a 6LoBLE interface with both global and link-local addresses, generated by SLAAC.

```
BLE IPv6 Node>ifconfig

Interface 0: 6LoBLE
        Link local address (LL64): fe80::204:9fff:fe00:b
        Global address: 2001:db8::204:9fff:fe00:b
```

```
BLE IPv6 Router>ifconfig

Interface 0: 6LoBLE
        Link local address (LL64): fe80::204:9fff:fe00:c
        Global address: 2001:db8::204:9fff:fe00:c
```

5.5. Multicast group

To verify the multicast groups that the devices have joined, enter *mcastgroup show*.

The IPv6 Node will have joined the all-nodes multicast address.

```
BLE IPv6 Node>mcastgroup show

Interface 0:
Multicast Group: ff02::1
```

The IPv6 Router will have joined the all-routers multicast address and the all-nodes multicast address.

```
BLE IPv6 Router>mcastgroup show

Interface 0:
Multicast Group: ff02::2
Multicast Group: ff02::1
```

5.6. Echo Ping

To verify all parameters for the echo ping command enter *help ping*:

```
BLE IPv6 Node>help ping
ping - IP Stack ping IPv4/IPv6 addresses
   ping <ip address> -i <timeout> -c <count> -s <size> -t <continuous ping> -S <source IP
address>
```

To send a ping from the IPv6 Node to the IPv6 Router, enter the IP address of the router:

```
BLE IPv6 Node>ping 2001:db8::204:9fff:fe00:c
Pinging 2001:db8::204:9fff:fe00:c with 32 bytes of data:
Reply from 2001:db8::204:9fff:fe00:c: bytes=32 time=80ms
Reply from 2001:db8::204:9fff:fe00:c: bytes=32 time=60ms
Reply from 2001:db8::204:9fff:fe00:c: bytes=32 time=60ms
Reply from 2001:db8::204:9fff:fe00:c: bytes=32 time=60ms
```

5.7. Echo UDP

The Echo UDP module is used to send test UDP data to a destination which echoes the payload back to the originator. To verify all parameters for the echo UDP command enter *help echoudp*:

```
BLE IPv6 Node>help echoudp
echoudp - Echo udp client
   echoudp -s<size> -S<source address> -t<continuous request> -i<timeout> <target ip
address>
```

To send an echo UDP from the IPv6 Node to the IPv6 Router, enter the IP address of the router:

```
BLE IPv6 Node>echoudp 2001:db8::204:9fff:fe00:c
Message sent to 2001:db8::204:9fff:fe00:cw with 32 bytes of dat AMessage received from
2001:db8::204:9fff:fe00:c: bytes=32, time68ms
```

5.8. Sockets

The shell commands available for the socket demo application are displayed after typing *help socket* in console:

```
BLE IPv6 Node>help socket
socket - IP Stack BSD Sockets commands
   socket open <protocol> <remote ip addr> <remote port>
   socket send <socket id> <payload>
   socket close <socket id>
```

These commands are valid for opening a socket as a client. In this example, at the Ipv6 Node, an UDP socket is opened and connected to the remote address 2001:db8::204:9fff:fe00:c, port 1234. Port 1234 is created by default on both devices:

```
BLE IPv6 Node>socket open udp 2001:db8::204:9fff:fe00:c 1234 6
Opening Socket... OK
Socket id is: 0
```

To send data enter the socket id and the data value:

```
BLE IPv6 Node>socket send 0 HelloWorld!
Socket Data Sent
```

On the IPv6 Router, the console outputs the string received on the socket:

```
HelloWorld! From IPv6 Address: 2001:db8::204:9fff:fe00:b
```

To close the socket on the IPv6 Node, enter the socket id:

```
BLE IPv6 Node>socket close 0
Socket 0 was closed
```

5.9. CoAP

The example sends a packet over the air using CoAP. When sending a CoAP message, all command parameters are mandatory. One has to specify the type of the message CON (confirmable message, waits for ACK) or NON (does not wait for any ACK), the request code, the IPv6 destination address, the URI-path options included in the message, and payload.

NOTE: URI-path options must be separated by /. Also the / separator is expected before the first URI-path option.

The shell commands available for the socket demo application are displayed after typing *help coap* in console:

```
BLE IPv6 Node>help coap
coap - Send CoAP message
   coap <reqtype: CON/NON> <reqcode (GET/POST/PUT/DELETE)> <IP addr dest> <URI path>
<payload ASCII>
Example: coap CON POST 2001::1 /led on
         coap CON POST 2001::1 /led off
         coap CON POST 2001::1 /led toggle
         coap CON POST 2001::1 /led flash
         coap CON POST 2001::1 /led rgb r255 g255 b255
```

To send a CoAP POST message form the IPv6 Node to the IPv6 Router, enter the following:

```
BLE IPv6 Node>coap CON POST 2001:db8::204:9fff:fe00:c /led flash
coap rsp from 2001:db8::204:9fff:fe00:c ACK
```

The RGB LED on the IPv6 Router should start flashing. If no ACK for the CoAP message is received after 4 retransmissions, an error message is displayed.

Revision history

Table 1. Revision history

Revision number	Date	Substantive changes
0	09/2016	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: BLEIP6UG
Rev. 0
09/2016