UG116: Developing Custom Border Router Applications

A Thread Border Router is a device in a Thread network that provides connectivity to adjacent IP networks on other physical layers. The Border Router provides services for devices within the Thread network, including routing services for off-network operations. This user’s guide provides information for building a border router as well as additional detail for those who are developing custom Border Router applications.

The instructions in this document are intended for use with Silicon Labs Thread SDK version 2.10.0 GA, or later.

KEY POINTS
- Development environment
- Building and installing the components of a border router system
- Commissioning and testing
- Detailed description of the Border Router operation
- Details on the configuration changes made by the border router package
- Additional information on utilities and resources
1. Introduction

A border router is an essential component of a Thread network that manages the traffic between the Thread network and adjacent IP networks. A border router facilitates a number of key capabilities including the following:

- Commissioning of Thread nodes
- GUA (Global Unicast Address) or ULA (Unique Local Address) assignment
- IP routing between the Thread network and adjacent IP networks
- Backbone support to enable Thread domains that integrate multiple Thread networks

This user guide is intended for software engineers who wish to develop a Thread Border Router System, and who are familiar with the Thread specification, Silicon Labs Thread stack and Silicon Labs development kits.

This user guide assumes familiarity with these documents:

- QSG113: Getting Started with Silicon Labs Thread
- UG103.11: Thread Fundamentals
- AN1198: Using Thread 1.2 Commercial Support Features
- AN1010: Building a Customized NCP Application
- UG278: Zigbee Cluster Library over IP User’s Guide

- thread-framework-readme.txt (available with the Silicon Labs Thread stack distribution)

**Note:** When discussing commissioning in this document, we only refer to Thread 1.1 commissioning, as Thread 1.2 commissioning is still under draft and review from the Thread board, and we do not have the means to commission Thread 1.2 devices with an external registrar, as described in the Thread spec and UG103.11: Thread Fundamentals. Ignore any references to a Thread 1.1 commissioner if it is not relevant to your use case.

After an introduction discussing the border router architecture, this user guide contains two major sections:

- **Border Router Implementation**: Instructions on how to get started with a border router implementation.
  - **Set Up the Border Router Development Environment**: How to obtain, connect, and configure the elements of a Thread system, including a Raspberry Pi-based border router.
  - **Build the Border Router Software Components**: How to build the border router’s software components with the current Silicon Labs Thread SDK, and update the initial border router configuration with them.
  - **Demonstrate the Thread Border Router**: How to get a Thread system up and running, including commissioning end nodes with an Android device.

- **Border Router Reference**: Information on a variety of topics including:
  - **Operation**: Describes the operation of the Thread Border Router System, including commissioning, discovery and end-to-end IPv6 communication.
  - **Border Router Configuration**: Provides a reference to explain the modifications the silabs-border-router package installation makes to the Raspbian Stretch Lite operating system installation.
  - **Utilities, Files and Services**: Contains a listing that may be useful for Border Router configuration and development.
  - **Resources**: Includes additional information available for software engineers.
2. Border Router Architecture

The following figure illustrates the key components of a border router configuration:

- A Thread 1.1 Commissioner, in this case an Android device.
- A Host, installed on a Raspberry Pi, consisting of three services:
  - ip-driver-app
  - commission-proxy-app
  - border-router-mgmt-app
- An NCP host, installed on a WSTK. While other NCP applications can be used, we recommend that you begin development with ncp-uart-hw.
- End nodes. In this document the end node application is sensor-actuator.

These elements are described in more detail in the subsequent sections.

2.1 Thread 1.1 Commissioner App for Android

The Thread commissioner app for Android is a utility for commissioning Thread 1.1 devices. It exchanges commissioner data via Wi-Fi with the commission-proxy-app on the Thread Border Router host and features a tool to securely commission Thread end nodes on the 802.15.4 Thread network. More information about commissioning can be found in the Thread Commissioning white paper, which is publicly available from the Thread Group.

The external Thread 1.1 commissioner app and source for Android is provided by the Thread Group and can be acquired as described in section 3.1.10 Install the Thread 1.1 Commissioner for Android. The steps for building the application are outside the scope of this document.
2.2 IP Driver Application (ip-driver-app)

The IP driver application (ip-driver-app) talks to the NCP application (over UART), in effect communicating with the stack on the Thread network. It acts as a multiplexer for various types of messages. Data received or sourced from the NCP is handed off to the ip-driver-app. If the data is commissioning data (including unsecure DTLS data), that is sent to the commission-proxy-app. If the data is management packets (TMSP) or regular mesh data that the border router needs to process, then it is sent to the border-router-mgmt-app. Other data that needs processing by the IP stack is sent directly to the stack.

2.3 Commissioning Proxy (commission-proxy-app)

The Border Router commissioner proxy (commission-proxy-app) communicates through sockets to the processes on the host, and over WiFi/MDNS to a 1.1 external commissioner. It proxies packets to an external commissioner so it can commission a node onto a network. It also delivers commissioner packets (petitions, acknowledgements, other management packets) to the leader or border router. It is composed of three pieces: Security code (currently only JPAKE), commissioning code, and CoAP. It manages the communication between the Thread stack and the external commissioner.

2.4 Border Router Management Application (border-router-mgmt-app)

The Border Router management application (border-router-mgmt-app) is a sample application on the host that performs various management and API calls so that it can act as a Thread border router. It is composed of a state machine that can automatically put the border router on the Thread network and enable other border-router functionalities on the host, including address configuration, network route management, ND/NS, service discovery, and so on. Its features include:

• Thread network configuration.
• Device discovery.
• Reacting to changes in network state.
• Removal of Thread end nodes from the Thread network.
• Removal of the Thread Border Router from the Thread network.
• Backbone link functionality by enabling a separate backbone support plugin, as described in AN1198: Using Thread 1.2 Commercial Support Features.

2.5 NCP Application (ncp-uart-hw)

The IP modem application (in this document ncp-uart-hw) on the Border Router runs the Thread stack software. Its features include a serial host interface that can run on Silicon Labs devices such as EFR32MG.

2.6 Thread End Node Application (sensor-actuator)

The Thread end node application in this document is sensor-actuator. It provides I/O for sensors, buttons and indicators. It exchanges data and management packets via the Thread network with the ncp-uart-hw application on the Border Router NCP. Its features include:

• Secure commissioning by the Thread commissioner app for Android.
• LED actuator control.
• Push button and temperature telemetry reporting.

2.7 Application

The application refers to an Android mobile device app, web browser plug-in such as Copper for the Mozilla browser, or any other application capable of communicating with devices on the Thread network via CoAP. The zclip-cli utility is a ZCL-over-IP implementation written in JavaScript that comes installed on the Border Router and is available at https://github.com/SiliconLabs/zclip-cli. It provides a library useful for interacting with devices, scripting and off-mesh development platforms like cloud and mobile. It comes with a CLI interface that has been added to SPATH on the Border Router facilitating device discovery and initialization. It may also be installed on any computer, such as the host computer, with an IPv6 connection to the Border Router. Demonstration with zclip-cli is described in UG278: Zigbee Cluster Library over IP User’s Guide.
3. Border Router Implementation

This chapter contains the following sections:

- **Set Up the Border Router Development Environment**: How to obtain, connect, and configure the elements of a Thread system, including a Raspberry Pi-based border router.

- **Build the Border Router Software Components**: How to build the border router’s software components with the current Thread SDK, and update the initial border router configuration with them.

- **Demonstrate the Thread Border Router**: How to get a Thread system up and running, including commissioning end nodes with an Android device.

3.1 Set Up the Border Router Development Environment

This section describes how to obtain, connect, and configure the elements of a Thread system, including a Raspberry Pi-based border router.

A typical Thread Border Router setup, shown in the following figure, includes the following components:

- Thread Border Router with Wi-Fi Access Point
- Thread End Devices such as EFR32 Mighty Gecko Wireless SoC Starter Kit
- Host computer with Simplicity Studio and SSH client for accessing the Border Router
- Commissioning device such as an Android handset
- IPv6 Ethernet Connectivity

![Figure 3.1. Thread System Components](image-url)
3.1.1 Order and Register the EFR32 Mighty Gecko Wireless SoC Starter Kit (WSTK)

Kits may be ordered from http://www.silabs.com/products/wireless/mesh-networking/zigbee/Pages/zigbee.aspx. The EFR32 Mighty Gecko Wireless SoC Starter Kit includes the following:

- 3 x Wireless starter kit mainboard
- 3 x EFR32MG12 2.4 GHz 19.5 dBm radio board
- 3 x EFR32MG12 2.4 GHz 10 dBm radio board
- AA Battery Board (supports running +19.5 dBm from battery)
- Cables
- EFR32MG Getting Started Card

Registration allows access to the support portal, and is a necessary step to access required software and documentation. The instructions that come with the EFR32 Mighty Gecko Wireless SoC Starter Kit include a unique registration key and explain the registration procedure.

3.1.2 Install Simplicity Studio and the Thread Software Development Kit on the Host Computer

Simplicity Studio is required to build several components of the Thread border router system as well as to flash Thread end node firmware. Refer to QSG113: Getting Started with Silicon Labs Thread, which can be found under the Getting Started section of the Simplicity Studio launcher perspective, for a detailed tutorial. The tutorial includes WSTK installation and configuration instructions, and describes the process for compiling and running an example for EFR32MG devices.

3.1.3 Order a Raspberry Pi

Refer to https://www.raspberrypi.org for recommended vendors for each component.

- Raspberry Pi 2 Model B, Raspberry Pi 3 Model B, or Raspberry Pi 3 Model B+
- 16-32 GB MicroSD card
- Edimax EW-7811UN USB 2.0 Wireless Adapter (required for Raspberry Pi 2 Model B only)
- Power supply

3.1.4 Set up the Raspberry Pi

1. Connect the USB Wi-Fi Adapter to one of the Raspberry Pi's USB ports.
2. Connect the Raspberry Pi's Ethernet port to the Internet with an Ethernet cable.

Note that it may be desirable to use an Ethernet switch as shown in Figure 3.1 Thread System Components on page 5, however, it is not necessary that the Raspberry Pi be on the same Ethernet network as the host computer and Wireless Starter Kit mainboards.

3. Connect one of the Wireless Starter Kit mainboards to the Raspberry Pi with a USB cable. This will become the network co-processor as shown in Figure 3.1 Thread System Components on page 5.
4. Connect a monitor to the HDMI port and a keyboard to a free USB port.
5. Plug in the Thread Border Router power supply.

Note: The Raspberry Pi and NCP Wireless Starter Kit are collectively referred to as the Border Router.

3.1.5 Set Up WSTK Hardware

Connect each wireless starter kit mainboard and the host computer as shown in Figure 3.1 Thread System Components on page 5. These connections will permit programming and network analysis of the NCP and end devices. Optionally, end devices may be connected to the host computer by USB rather than Ethernet.

3.1.6 Etch the MicroSD card for the Raspberry Pi

Use a host computer to install the Raspbian Stretch Lite operating system on the SD card as described here: https://www.raspberry-pi.org/downloads/raspbian/

Install the SD card in Raspberry Pi and power it on.

Note: Only use the Stretch Lite operating system. Other versions are not supported.
3.1.7 Locate the Raspberry Pi and Enable SSH

You must know the Raspberry Pi’s IP address and enable SSH on it for some of the procedures in this chapter. Configure the Raspberry Pi with an IP address (preferably static). If the Raspberry Pi can only acquire an IP address from a DHCP server you will need to find it on the network and obtain its IP address.

Log in to the Raspberry Pi. The default username is “pi” and password is “raspberry.” On first reboot you are prompted to change the password. Configure the default keyboard layout and enable the SSH server with:

```bash
$ sudo raspi-config
```

1. Select Interfacing Options.
2. Navigate to and select SSH.
3. Select Yes.
4. Select OK.
5. Select Finish.

3.1.8 Connect the Host Computer to the Raspberry Pi

Establish an SSH connection to the Border Router using your choice of an SSH client.

3.1.9 Install the Border Router Software

To configure the border router with the software infrastructure and stub versions of the software components, run the following shell script or enter the commands manually in the same order. You must then replace the applications installed by this procedure with current builds as described in section 3.2 Build the Border Router Software Components.

```
#!/bin/bash
# Convert Raspbian (Stretch) to a Border-Router
if [ "$1" == "" ]; then
  echo "Usage: stagePi.sh <rpi IP address>"
  exit
fi
RPI_HOST="pi@$1"
# Copy our public key to allow passwordless logins to the BR
ssh-copy-id -i ~/.ssh/id_rsa.pub ${RPI_HOST}
# ssh -CX ${RPI_HOST} "sudo raspi-config"
# Install BR software
ssh ${RPI_HOST} "echo 'deb http://devtools.silabs.com/solutions/apt stretch main' | sudo tee --append /etc/apt/sources.list"
ssh ${RPI_HOST} "sudo apt-get update"
ssh ${RPI_HOST} "sudo apt-get -y install dirmngr && sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 90CE4F77 && sudo apt-get update"
ssh ${RPI_HOST} "sudo apt-get -y install silabs-border-router vim tcpdump dnsutils conntrack libncurses-5-dev libreadline-dev npm && sudo apt-get autoremove"
# Cleanup
ssh ${RPI_HOST} "sudo rm /var/cache/apt/archives/*.deb -f"
# Speed up compiles to use multiple processors and better scheduling policy
ssh ${RPI_HOST} "echo "time chrt --batch 0 make --silent -j4"' >> ~/.bashrc"
# Remember to reboot the rPi!
```

For Thread 1.2 off-mesh multicast support, a multicast routing daemon must be installed on the border router. Follow this additional step to install SMCRoute, which is the program we support for this purpose. It is not available via apt-get but it is easy to install:

1. Download SMCRoute: https://github.com/troglobit/smcroute/releases/download/2.4.4/smcroute-2.4.4.tar.gz
2. Deploy the tar anywhere on the pi, then run the following commands for a quick installation (also in the included README):

```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var
make -j5
sudo make install-strip
# After installation
sudo reboot
```
3.1.10 Install the Thread 1.1 Commissioner for Android

An Android handset or tablet with the Thread Group commissioning app is required to commission Thread end devices. The Thread Group commissioning app is distributed in the Google Play store and is also distributed in the Thread Border Router file system. The file may be transferred from the Border Router to a host with a utility such as WinSCP or SCP.

To access the source for the commissioning app, refer to the Thread Group Bitbucket repository https://bitbucket.org/threadgroup/, and request access to the repository through the Thread Group.

<table>
<thead>
<tr>
<th>Device</th>
<th>Programming File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android 5.0+</td>
<td>/opt/siliconlabs/threadborderrouter/commissioning/thread-commissioning-app-1.01.11.apk</td>
</tr>
</tbody>
</table>

3.2 Build the Border Router Software Components

This section describes how to update the default border router configuration with software components built with a current Thread SDK.

The best practice is to build the NCP bootloader, the NCP firmware, and all of the border router applications from the same Gecko SDK and Thread stack so that each component is on the same version.

For details on building an application in Simplicity Studio, see QSG113: Getting Started with Silicon Labs Thread.

3.2.1 Build the Bootloader and NCP Firmware

This procedure loads software onto the WSTK connected to both the Raspberry Pi and the Ethernet. Launch Simplicity Studio. You should see your target board in the upper right Debug Adapters pane. To build and flash the bootloader:

1. On the launcher page click [New Project].
2. Click Gecko Bootloader, and click [Next].
3. If prompted, select the Gecko SDK Suite platform bootloader, and click [Next].
4. If you are using UART communication to the NCP (recommended), select UART XMODEM bootloader. For SPI select EZSP SPI Bootloader. Click [Next].
5. On Project Configuration, make any changes you want to make and click [Next].
6. On the Project Setup page, make sure you see a board and part number filled in based on your connected mainboard. If you have installed IAR and GCC, select your toolchain. GCC is set as active by default, but you can change the active toolchain to IAR. You can also change the toolchain once the project has opened.
7. If the NCP device is connected by Ethernet but is on a different subnet than your PC with Studio, you must configure its IP information. Select Preferences > Simplicity Studio > Device Manager > TCP/IP Adapters.
8. Click [Finish].
9. In the AppBuilder interface, click [Generate] in the upper right to generate additional project files. Note that any time you change a project configuration and save the changes, you must regenerate the project files.
10. Build and flash the bootloader to your NCP board. The UART bootloader’s default name is bootloader-uart-xmodem-combined.s37 and the SPI’s is bootloader-spi-ezsp-combined.s37.

To build and flash the NCP application, click the Launcher tab in the upper right to return to the Launcher perspective

1. On the launcher page click [New Project].
2. Click Silicon Labs Thread, and click [Next].
3. Select NCP-UART-HW as the NCP application type. Click [Next].
4. On Project Configuration, make any changes you want to make and click [Next].
5. On the Project Setup page, make sure you see a board and part number filled in based on your connected mainboard. If you have installed IAR and GCC, select your toolchain. GCC is set as active by default, but you can change the active toolchain to IAR. You can also change the toolchain once the project has opened.
6. Click [Finish].
7. In the AppBuilder interface, click [Generate] in the upper right to generate additional project files. Note that any time you change a project configuration and save the changes, you must regenerate the project files.
8. Build and flash the NCP application to your NCP board.
3.2.2 Build and Install End Device Applications

The remaining two WSTK mainboards should be configured as end devices. While any of the SoC examples should work, this document uses the Sensor/Actuator example application because an Android device can commission it based on a QR code (see section 3.3.5 Commission Thread 1.1 End Nodes with an Android Device).

You do not need to build the bootloader image again. Follow a similar procedure to that described for the NCP application above but select application type Silicon Labs Thread and sample application Sensor Actuator. You can flash the compiled application image along with the bootloader image you created in section 3.2.1 Build the Bootloader and NCP Firmware.

3.2.3 Build and Install Border Router Host Services

Before replacing any or all of the three border router host services, you should:

1. Stop all border router host applications:
   
   ```
   sudo service border-router-apps stop
   ```

2. Deploy the application(s) as described in the following sections. The instructions use the make optimization alias in .bashrc created by the shell script in section 3.1.9 Install the Border Router Software.

3. Start all applications.
   
   ```
   sudo service border-router-apps start
   ```

3.2.3.1 Build and Install the border-router-mgmt-app

To create the border-router-mgmt-app, first generate the source files to be compiled on the Raspberry Pi.

1. Open Simplicity Studio.
2. On the launcher page click [New Project].
3. Click Silicon Labs Thread, and click [Next].
4. Select Border Router Management App. Click [Next].
5. On Project Configuration, no changes are required. Click [Next]. Note: If Thread 1.2 backbone support is required, follow the instructions in AN1198: Using Thread 1.2 Commercial Support Features, and enable the bbr-support plugin as described.
7. Click [Finish].
8. In the AppBuilder interface, click [Generate] in the upper right to generate additional project files. Note that any time you change a project configuration and save the changes, you must regenerate the project files.

Next, compile the application on the Raspberry Pi.

1. Compile the code locally on your Raspberry Pi. To compile the project, issue the following command from the directory containing the .isc and the other generated files.
   
   ```
   makef NO_READLINE=1
   ```

   To locate these files from Studio, in Project Explorer pane right-click the .isc file and select Properties or select the .isc file and press alt-enter. The parent directory is listed on the Properties window.

   If you have any problems building, revisit Studio and re-create the project without any changes of your own.

   If the result of your build is not a file with the exact name border-router-mgmt-app, rename it to that exact name.
2. The existing application is located in /opt/siliconlabs/threadborderrouter/bin. Replace it with your newly-created application.
3. Check that the permissions of the new file match the permissions shown here. If they do not, adjust them accordingly.

   ```
   -rwxr-xr-x 1 pi pi 2338996 Jan 16 19:01 border-router-mgmt-app
   ```
3.2.3.2 Build and Install the ip-driver-app

To create the ip-driver-app:
1. Compile the code locally on your Raspberry Pi. To compile the project, issue the following command from the protocol/thread directory:

   ```
   make -f app/ip-ncp/ip-driver-app.mak
   ```

2. Once the build completes successfully, locate the executable file, `ip-driver-app`, in the Gecko SDK subdirectory `protocol/thread/build/ip-driver-app/`.

   If the result of your build was not a file with the exact name `ip-driver-app`, rename it to that exact name.

3. The existing application is located in `/opt/siliconlabs/threadborderrouter/bin`. Replace it with your newly-created application.
4. Check that the permissions of the new file match the permissions shown below. If they do not, adjust them accordingly.

   ```
   -rwxrwxr-x 1 pi pi 259128 Mar 8 2018 ip-driver-app
   ```

3.2.3.3 Build and Install the commission-proxy-app

To create the commission-proxy-app:
1. Compile the code locally on your Raspberry Pi. To compile the project, issue the following command from the protocol/thread directory:

   ```
   make -f app/thread-commissioning/commission-proxy-app.mak
   ```

2. Once the build completes successfully, locate the executable file, `commission-proxy-app`, in the Gecko SDK subdirectory `protocol/thread/build/commission-proxy-app/`.

   If the result of your build was not a file with the exact name `commission-proxy-app`, rename it to that exact name.

3. The existing application is located in `/opt/siliconlabs/threadborderrouter/bin`. Replace it with your newly created application.
4. Check that the permissions of the new file match the permissions shown below. If they do not, adjust them accordingly.

   ```
   -rwxrwxr-x 1 pi pi 769265 Mar 8 2018 commission-proxy-app
   ```

3.3 Demonstrate the Thread Border Router

This section describes how to bring up a Thread network with the updated border router created in the previous section, and then commission end nodes with an Android device.

3.3.1 Power On the Border Router

3.3.2 Connect the Host Computer to the Border Router

1. Connect to the Border Router Wi-Fi access point.
   - SSID: “Silicon Labs Thread XXXX”
   - Passphrase: “solutions”

2. Establish an SSH connection to the Border Router.
   - Username: `pi`
   - Password: `<as established during initial setup>`

   ```
   $ ssh pi@192.168.42.1
   ```
3.3.3 Reset the Thread Network

1. Send the network reset command to the Border Router.

```
$ echo 'network-management reset' > /dev/border-router-cln
```

2. Confirm border-router-mgmt-app is "up" by tailing its log.

```
$ tail -f /var/log/siliconlabs/border-router-mgmt-app
<22:04:17> Border router is up
```

3.3.4 Power On Each End Device

Each device has a 64-bit EUI and a fixed, eight-digit, base32 joining passphrase (also called the Joining Device Credential) derived from the EUI, both of which are required by the commissioning application. The joining passphrase and EUI print to stdout of the end device console on boot. Additionally, the EUI is available with the `info` command. These are needed in the next step. See QSG113: Getting Started with Silicon Labs Thread for guidance on using the end device console within Simplicity Studio.
3.3.5 Commission Thread 1.1 End Nodes with an Android Device

1. Using the device on which the Thread 1.1 commissioning app was installed in section 3.1.10 Install the Thread 1.1 Commissioner for Android, connect to the Thread Border Router Wi-Fi Access Point. The access point will have an SSID of the format “Silicon Labs Thread XXXX,” where “XXXX” is an arbitrary hex number generated randomly after installation of the Border Router Software.
   - SSID = “Silicon Labs Thread XXXX”
   - Passphrase = “solutions.”

2. Launch the commissioning app. The app will search for available Thread networks and request the Thread admin password. The Thread admin password is set at compile time by the Border Router application and printed on stdout immediately after boot. It is “COMMPW1234” for this version of the Thread Border Router.

3. If successful, the commissioning application will indicate that it is ready to accept join credentials by displaying a screen similar to the following:

4. To join each end device using the commissioning app:
   a. Enter the EUI and connect code (also called the Passphrase) for a Thread End Node as noted in section . The commissioning app should pop up a busy screen indicating that it is locating the Thread End Node to join.
   b. Press the Thread End Node PB1 to issue a net reset command and initiate a scan of all available channels for the Thread Network.
   c. The Thread End Node scans all channels twice and then will go idle unless the Network is found. It is important to enter the connect code on the app first and press Button1/PB1 second. A successful commissioning process is indicated in the commissioning app screen by a large checkmark and a “Device Added” message.

Note: After any network change, such as adding or removing a device or receiving a new IPv6 prefix, the Thread network may take up to two minutes to heal. Devices might not operate correctly during this time.

LED1 on the EFR32MG kits indicates the type of node by using different blink patterns:
   1. Very fast blink with pause (either on or off) = joining
   2. Fast blink = router
   3. Slow blink = end node
   4. Alternating fast and slow blink = leader

Hint: To save time typing in the EUI and Passphrase, create a QR code at http://goqr.me using a string formed from the version, joining passphrase phrase, and EUI. For example, the string

`v=1&cc=EL7TUCB0&eui=000B57FFFE07F6B3`
is formed from Version v=1, joining passphrase phrase cc=EL7TUCB0, and EUI=000B57FFFE07F6B3. Successful generation of this example is shown in the following figure. You can then use the Android device to scan the QR codes specific to your End Nodes.

![QR Code Example](image.png)

**3.4 Troubleshooting**

Debugging the Border Router often requires that the applications allow interaction. This section describes how to run the Border Router applications in the foreground and details some of the “best practices” when trying to do things manually.

**3.4.1 Disable Border Router Application Auto-Startup**

Running these two commands on your Raspberry Pi should disable automatic startup of the border router applications on subsequent reboots:

```bash
sudo update-rc.d border-router-apps disable
sudo systemctl daemon-reload
```

To re-enable automatic startup, use:

```bash
sudo update-rc.d border-router-apps enable
sudo systemctl daemon-reload
```

It is possible that disabling automatic startup does not work completely. To see if any border router applications are running in the background, execute the following commands. You will need to kill any processes associated with the `ip-driver-app`, `border-router-mgmt-app`, and `commission-proxy-app`. You should also make sure that silabs-scripts is not running. If that process is running, you will also need to kill that as well.

```
ps -ef | grep app
ps -ef | grep silabs-scripts
```
3.4.2 Manually Launching Border Router Applications

You can use the silabs-scripts executable to launch the border router applications. To bring up just the border-router-mgmt-app in the foreground with the ip-driver-app in the background, use:

```
sudo /opt/siliconlabs/threadborderrouter/bin/silabs-scripts -i bg -b fg
```

To bring up the commission-proxy-app in the background as well, use:

```
sudo /opt/siliconlabs/threadborderrouter/bin/silabs-scripts -i bg -c bg -b fg
```

Launching in this way allows you to interact with the application you select for the foreground (fg) while running the remaining applications in the background. If you want to have all processes in the foreground, you can run each command in its own window. Note that the port values in these examples (4901, 8888) are examples. We recommend these defaults, but if you have local services that create a port collision you can change them as needed.

```
sudo /opt/siliconlabs/threadborderrouter/bin/ip-driver-app -u /dev/ttyUSB0 -t tun0 -m 4901 -c 8888
sudo /opt/siliconlabs/threadborderrouter/bin/thread-comm-app --port 49191 --mgmt_port 8888
sudo /opt/siliconlabs/threadborderrouter/bin/border-router -m 4901
```

Alternatively, you can run all the apps in the background and simply tail the associated log files, located in /var/logs/siliconlabs.

3.4.3 Form the Network Manually

First, disable automatic network formation.

1. Edit /etc/border-router.conf.
2. Set AUTO_FORM_NETWORK to 0.
3. In the foreground console of the border-router-mgmt-app, type "network-management reset".

Next, halt all devices on the network to ensure that the network is formed using your parameters. If these devices are Silicon Labs devices, the simplest way to do this is to erase any application running on these devices. Depending on the application, other methods may be available.

To form the network, in the foreground console of the border-router-mgmt-app, enter a command with the following format:

```
network-management commission <preferred channel (in decimal)> <fallback channel mask> <network name> <ula prefix> <extended pan id*> <master key> <pan id>
```

Example:

```
network-management  commission 11 0 "testnetwork3333" "FD01::/64" {7777C75FB4E1EFC6} {656D62657220454D3235302063680102} 0x3333
```
4. Border Router Reference

This chapter covers the following information:

- Detailed walkthrough of the border router's operation.
- Description of the configuration changes made by the border router package.
- Utilities, files, and services of use to developers.
- Other resources, including training and documentation.

4.1 Operation

This section describes the operation of the Thread Border Router System, including commissioning, discovery and end-to-end IPv6 communication.
4.1.1 Understanding the Border Router Management Application

The Thread Border Router Management Application (border-router-mgmt-app) orchestrates much of what happens during network formation and in setting the parameters for use in a new Thread network. Throughout the remainder of this chapter the term Border Router and Border Router Management Application will be used interchangeably. The state machine for the Border Router Management application is shown in the following figure.

---

**Network state**
(Network Layer Status)

- **EMBER_SAVED_NETWORK**
- **EMBER_NO_NETWORK**
- **EMBER.Joining_NETWORK**
- **EMBER_Joined_NETWORK Attaching**
- **EMBER_Joined_NETWORK Attached**

---

**Border-router state**
(Application Layer Status)

- **RESET_NETWORK_STATE**
- **RESUME_NETWORK**
- **FORM_NETWORK**
- **JOIN_NETWORK**
- **GET_COMMISSIONER**
- **CONFIGURE_BORDER_ROUTER**
- **INITIALIZE_BORDER_ROUTER**
- **UP_STATE**

---

**Figure 4.1. Border Router Management Application State Diagram**

EMBER_Joined_NETWORK Attached is the network state that corresponds to full connectivity at the Network Layer. The application layer then performs higher-level configuration such as setting up a commissioner and setting up DNS.
The left half of the state machine drawing, represented by grey-filled rectangles, corresponds to the status of the network layer. The Silicon Labs Thread stack drives this state machine, and the Border Router has no direct control over it. The Border Router does indirectly influence the network state by calling various stack functions. These stack functions that influence the network state are shown in labels above some of the arrows drawn from the border-router state machine to the network state diagram. Note that the network state diagram is incomplete and only shows states of interest to the Border Router. The right half of the figure describes the state of the Border Router. Each state is briefly described here:

**RESET_NETWORK_STATE**: In this state the Border Router calls stack function `emberResetNetworkState`. This clears network information from the NCP and immediately removes the Border Router from any network. User CLI command `network-management reset` also calls the network stack function `emberResetNetworkState`.

**RESUME_NETWORK**: The border router always transitions to this state if the stack transitions to the EMBER_SAVED_NETWORK state. It calls the stack function `emberResumeNetworkState`, which causes the stack to attempt to re-join a previously-known network. This is usually what will happen if the border-router is rebooted during the UP_STATE.

**FORM_OR_JOIN_NETWORK**: The Border Router transitions to this state if the stack transitions to the EMBER_NO_NETWORK state. If `AUTO_FORM_NETWORK=1` the Border Router moves to the FORM_NETWORK state, and if `AUTO_FORM_NETWORK=0` the Border Router waits on CLI input from the user. If the user issues the `border-router start-connect` command, the Border Router moves to the JOIN_NETWORK state. If the user issues the `network-management commission` command, the Border Router moves to the JOIN_NETWORK_COMPLETION state.

**FORM_NETWORK**: In this state the Border Router calls the stack function `emberFormNetwork` to form a network with automatically-generated network settings.

**JOIN_NETWORK**: In this state the Border Router calls the stack function `emberJoinNetwork` to perform an active scan for joinable networks. An external in-band commissioning utility such as the Thread Group Commissioning App is required to complete the joining process.

**JOIN_NETWORK_COMPLETION**: In this state the Border Router calls the stack function `emberJoinComissioned` and joins a pre-defined network by out-of-band injection of network settings such as the Network Master Key. This is especially useful for debugging when packet decryption is needed.

**GET_COMMISSIONER**: This state is always reached when the stack transitions to EMBER_JOINED_NETWORK_ATTACHED. The transition to that network state means that the Border Router can communicate as a member of a Thread network, but has not yet configured other settings appropriate for a Border Router. This state sets the commissioner pre-shared key.

**CONFIGURE_BACKBONE_ROUTER**: (not shown) If the bbr-support plugin is enabled for Thread 1.2 Backbone Router support, this state is reached next. This allows the border router to set up and advertise the domain prefix in its Thread network, as well as initialize various functions of its backbone link.

**CONFIGURE_BORDER_ROUTER**: This state is reached upon successful completion of `emberGetCommissioner` or `emberBackbonePrefixHandler` (if bbr-support is enabled). It informs the stack of some of the services available, such as routing, DNS (domain name system) and SLAAC (stateless autoconfiguration) using a call to the stack function `emberConfigureGateway`.

**INITIALIZE_BORDER_ROUTER**: This state is reached upon a successful call to `emberConfigureGateway`. It does some final configuration and sets up DNS.

**UP_STATE**: The border-router should remain in this state as long as it is on a stable network and the user takes no action otherwise. A heartbeat is issued periodically in this state.
4.1.2 Border Router Network Form and Join

To form a network means the Border Router erases any existing network settings, such as the Network Master Key and extended PAN ID, and generates new ones. The use case for forming a network is when the Thread network does not already exist and knowledge of the Network Master Key is not required. The Border Router supports a procedure to form a network by automatically generating network settings.

Forming a network with automatically-generated network settings is the most secure method because a random Network Master Key will be generated and may not be retrieved. When end devices are commissioned the Network Master Key is shared with them over a secure DTLS connection. The Network Master Key is not shared with the commissioner and cannot be intercepted by a third party. Commissioning of end devices on a network with automatically-generated network settings is done with an in-band utility such as the Thread Group Commissioning Application.

To join a network means the Border Router erases any existing network settings, such as the Network Master Key and extended PAN ID, and replaces them with new settings. The Border Router supports two methods for joining a network: In-band joining with a utility such as the Thread Group Commissioning Application, and out-of-band joining by manual injection of network settings. In the case of out-of-band joining, it is convenient to think of the process as joining a network with pre-defined network settings, and that network need not be in radio range, or even physically exist. The use case for in-band joining is to join a Thread network on which a border agent and commissioner exist, and knowledge of the Network Master Key is either not required or is available by other means. The use case for out-of-band joining is to join a pre-defined Thread network on which a border agent and commissioner need not exist, or knowledge of the Network Master Key is required but is unavailable by other means.

Knowledge of the Network Master Key presents some risk, however, there are two reasons it would be required:

- **Debugging using packet trace**: Packet capture tools such as the Silicon Labs Network Analyzer require the Network Master Key to decrypt network traffic.
- **Out-of-band commissioning devices**: A known Network Master Key can be pre-shared with devices so that they can join a Thread network without going through the in-band commissioning process. Note that this is the only way that applications other than sensor-actuator can join a Thread network.

If the Border Router is on a network at boot it will resume on that network, otherwise the state of AUTO_FORM_NETWORK determines whether Border Router forms or joins a network:

- **AUTO_FORM_NETWORK = ON**: Automatically generates network settings and forms a network.
- **AUTO_FORM_NETWORK = OFF**: Waits for user input on the CLI to determine the joining method:
  - **In-band**: CLI command `border-router start-connect`
  - **Out-of-band**: CLI command `network-management commission`

The state of AUTO_FORM_NETWORK may be set in two ways:

1. In the `/etc/siliconlabs/border-router.conf` file. The default setting is AUTO_FORM_NETWORK=1 (on).

The following sections provide additional details for each of the three use cases:

1. **Network Form with Automatically-Generated Network Settings**
2. **Network Join with In-Band Commissioning**
3. **Network Join with Out-of-Band Commissioning**
4.1.2.1 Network Form with Automatically-Generated Network Settings

Network form with automatically-generated network settings is the default configuration on boot. The state diagram for this operation is shown in the following figure. See section to commission end devices onto this network.

**Figure 4.2. Network Form with Automatically-Generated Network Settings**
4.1.2.2 Network Join with In-Band Commissioning (Thread 1.1)

The state diagram for this operation is shown in the following figure. See section to commission end devices onto this network.

**Network state**
(Network Layer Status)

**Border-router state**
(Application Layer Status)

![Network Join with In-Band Commissioning Diagram](image)

EMBER_JOINED_NETWORK_ATTACHED is the network state that corresponds to full connectivity at the Network Layer. The application layer then performs higher-level configuration such as setting up a commissioner and setting up DNS.

**Figure 4.3. Network Join with In-Band Commissioning**
4.1.2.3 Network Join with Out-of-Band Commissioning

The state diagram for this operation is shown in the following figure. See section 4.1.3 End Device Out-of-Band Commissioning with Manual Injection of Network Settings.

Two methods may be used to commission end devices onto this network:

1. To use an external commissioner, go to section . Note that even though the network parameters are specified manually, the origin of the parameters is unknown both to the Thread network and to the external commissioner. Therefore, they behave the same in either case.

2. To provide network parameters to a Joiner device using the device CLI, go to section 4.1.3 End Device Out-of-Band Commissioning with Manual Injection of Network Settings. This method can only be used if the Network Master Key is known.

---

Figure 4.4. Network Join with Out-of-Band Commissioning
4.1.3 End Device Out-of-Band Commissioning with Manual Injection of Network Settings

If a network was formed using manually generated settings, those settings may be injected into another device to allow it onto the Thread network directly. This can be done from the console of an end device using the connection manager plugin, such as the sensor-actuator, with the following commands:

```
sensor-actuator-node> network-management reset
sensor-actuator-node> network-management commission 13 0 "example-id" "fd01::/64" (0102030405060708) (656D62657220454D3235302063686970) 1234 0
sensor-actuator-node> network-management join-commissioned 3 2 1
```

The `network-management commission` CLI takes the following parameters:

```
network-management commission <preferred channel:1> <fallback channel mask:4> <network id:0--16> <ula prefix> <extended pan id:8> <key:16> [<pan id:2> [<key sequence:4>]]
```

- Channel 0 means any channel.
- PAN id 0xFFFF means any PAN id.
- PAN id and key sequence are optional and may be omitted. If key sequence is specified, PAN id must be specified too, but may be 0xFFFF.

For example:

```
network-management commission 0 0 "example-id" "fd31:4159:2653:5897::/64" {0102030405060708} {656D62657220454D3235302063686970}
```

commissions the stack to join network "example-id" with ULA "fd31..." and extended PAN id {01...} on any channel using key {65...}. The parameters must be identical to the parameters of the commission command used in the `border-router-mgmt-app` CLI in order to put the border-router and the sensor-actuator on the same network.

4.1.4 Discovery

1. An application sends a multicast discovery request to the multicast address ff03::1.
2. Each Thread end node responds to the multicast with its global IPv6 address and device type.

![Figure 4.5. Discovery](image)

**Figure 4.5. Discovery**
4.1.5 End-to-End IPv6 Communication

An application such as the Copper plugin for Mozilla or zlicp.js (available on the Silicon Labs github repository https://github.com/SiliconLabs/zclp.js) sends and receives messages via CoAP to and from the IPv6 address of the desired Thread end node. This assumes commissioning and discovery have completed.

![Figure 4.6. End-to-End IPv6 Communication](image)

4.2 Silicon Labs Thread Border Router Configuration

The silabs-border-router package is a collection of files describing Silicon Labs host application binaries, support scripts that enable host applications and Linux utilities to interact, and modifications to common Linux utilities. Its installation configures the Raspbian Stretch Lite operating system to behave as a Border Router between the Thread and adjacent IPv6 networks. This section provides a reference to the modifications the silabs-border-router package makes to the Raspbian Stretch Lite operating system.

The following directories contain important files:

- All logs for the border router background processes are located in the directory /var/log/siliconlabs/
- Binaries are located in /opt/siliconlabs/threadborderrouter/bin
- Configuration is located in /etc/siliconlabs/border-router.conf (as well as within the sample application itself, which you can alter in Studio)
4.2.1 The Border Router Configuration File

System variables that either modify multiple daemons on the Border Router or apply directly to configuration of the Silabs Host Applications are installed into the Border Router Configuration file (/etc/siliconlabs/border-router.conf). This file includes network settings, host application settings, and the ability to enable and disable advanced router features such as DNS64 or NAT64. The file is formatted as a list of key-value pairs and includes comments detailing the behavior of each key-value pair. The following table describes the default key-value pairs that the border-router.conf file provides.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_VERSION</td>
<td>1</td>
<td>The current version of the Border Router Configuration File.</td>
</tr>
<tr>
<td>AUTO_FORM_NETWORK</td>
<td>1</td>
<td>A Boolean describing if the Border Router should automatically form a Thread network on startup using default settings, or if it should wait for the user to enter manual settings.</td>
</tr>
<tr>
<td>NETWORK_ID</td>
<td>br-XXXXX</td>
<td>A string providing a pseudorandom Thread network ID.</td>
</tr>
<tr>
<td>MESH_SUBNET</td>
<td>2001:db8:385:9318::/64</td>
<td>An IPv6 prefix and subnet width assigned to the Thread network (tun0).</td>
</tr>
<tr>
<td>HOST_SUBNET</td>
<td>2001:db8:8569:b2b1::/64</td>
<td>An IPv6 prefix and subnet width assigned to the Wi-Fi network (wlan0).</td>
</tr>
<tr>
<td>DOMAIN_SUBNET</td>
<td>2001:db8:385:93bb::/64</td>
<td>An IPv6 prefix and subnet width assigned to the Thread domain that connects multiple backbone border routers (default: eth0).</td>
</tr>
<tr>
<td>USE_PREFIX_DELEGATION</td>
<td>0</td>
<td>A Boolean describing if the Border Router should enable IPv6 Prefix Delegation on the Ethernet interface (eth0). Prefix delegation causes the MESH and HOST obtain dynamic prefixes from any prefix delegation server running on eth0.</td>
</tr>
<tr>
<td>NAT64_PREFIX</td>
<td>fc01:6464::/96</td>
<td>An IPv6 prefix whose subnet width must be exactly /96 used by the NAT64 and DNS64 services to map IPv4-only destinations to an IPv6 address.</td>
</tr>
</tbody>
</table>

4.2.2 Silicon Labs Host Applications

The package installs the three host applications: `ip-driver-app`, `commission-proxy-app`, and `border-router-mgmt-app`. The host applications are installed in the 'bin' folder within the SILABS_BORDER_ROUTER_ROOT directory: /opt/siliconlabs/threadborderrouter/.

4.2.3 Linux Configuration File Modifications

The `silabs-border-router` package has dependencies on a number of common Linux utilities. The utilities in the `silabs-border-router` package have been adapted to provide the functionality of a standard router through modifications to their configuration files. The `silabs-border-router` package modifies standard Linux configuration in one of four different ways: installation of new configuration files, automated injection of minor additional code into existing static configuration files, install-time complete replacement of existing static configuration files, and by using a template to dynamically rewrite existing configurations as the network state changes. In all files where configuration settings are required by the `silabs-border-router` package, comment sections between #SILABS_BORDER_ROUTER markers provide detailed information about each modification’s function and purpose, and provide hints for safely customizing the option in question.
4.2.4 New Configuration Files

In instances where a Linux utility does not provide its own default configuration, the silabs-border-router package installs a new configuration file to the appropriate directory within the Linux file system. All entries in these files should be considered ‘critical’ to the Border Router Add On Kit’s default functionality, and modifying them may result in the failure of an intended feature or service. The following table describes the ‘Installed Configuration Files’ the silabs-border-router package provides.

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/default/isc-dhcp-server</td>
<td>Contains settings used by the initialization script for the DHCP service.</td>
</tr>
<tr>
<td>/etc/init.d/border-router-apps</td>
<td>The initialization script that daemonizes the Silabs host applications on-startup.</td>
</tr>
<tr>
<td>/etc/logrotate.d/siliconlabs</td>
<td>Adds the Silicon Labs logfiles to the logrotate daemon’s scheduler.</td>
</tr>
<tr>
<td>/etc/modules-load.d/silabs.conf</td>
<td>Instructs the Linux Kernel to load drivers required by the Border Router Add On Kit.</td>
</tr>
<tr>
<td>/etc/siliconlabs/border-router.conf</td>
<td>Defines common variables used to coordinate between the host applications and operating system.</td>
</tr>
<tr>
<td>/etc/sysctl.d/30-silabs.conf</td>
<td>System control settings with priority (30) that are executed to enable IP routing.</td>
</tr>
<tr>
<td>/etc/udev/rules.d/99-silabs.rules</td>
<td>Startup rules at lowest priority (99) for the NCP.</td>
</tr>
<tr>
<td>/etc/iptables.ipv4.nat</td>
<td>Adds NAT44 rules to enable IPv4 routing.</td>
</tr>
<tr>
<td>/etc/dhcpcd.exit-hook</td>
<td>A script run when Prefix Delegation occurs. This script reconfigures tun0 and wlan0 to use a delegated prefix for their subnet.</td>
</tr>
</tbody>
</table>

4.2.5 Additional Configuration Settings

In instances where only minor changes to a configuration file are required in order to enable a feature of the Border Router, the silabs-border-router package injects additional configuration settings between the text tags #SILABS_AUTOGEN_START and #SILABS_AUTOGEN_END. These tags denote that text between them is volatile and will be removed or overwritten if the silabs-border-router package is removed or upgraded, and that certain dynamic processes may also overwrite these sections during normal Border Router operation. The configuration outside of these markers can usually be easily modified without disabling the Border Router’s intended functionality. Additional configuration settings are recorded in files prepended with the keyword ‘additional’ followed by the filename they modify. For example: additional-dhcpcd.conf contains additional settings that will be injected into the dhcpcd.conf file, between #SILABS_AUTOGEN markers. These additional configuration settings appear in files installed into the SILABS_BORDER_ROUTER_ROOT directory, which is located at /opt/siliconlabs/threadborderrouter. The following table describes the ‘Minor Configuration Files’ to which the silabs-border-router package adds additional lines.

<table>
<thead>
<tr>
<th>Destination File</th>
<th>Source file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/bash.bashrc</td>
<td>additional-bash.bashrc</td>
<td>Adds the Silicon Labs binary directory to PATH.</td>
</tr>
<tr>
<td>/etc/resolvconf.conf</td>
<td>additional-resolvconf.conf</td>
<td>Installs the optional DNS and DNS64 services to the host-resolve path.</td>
</tr>
<tr>
<td>/etc/dhcpcd.conf</td>
<td>additional-dhcpcd.conf</td>
<td>Disables the dhcpv4 client on wireless interfaces, enables Prefix Delegation on eth0.</td>
</tr>
<tr>
<td>/etc/dhcpcd/dhcpcd.conf</td>
<td>additional-dhcpcd.conf</td>
<td>Configures the DHCP server daemon, which assigns addresses on wlan0.</td>
</tr>
</tbody>
</table>

In instances where most of a configuration file must be modified in order to enable a feature of the Border Router, the silabs-border-router package backs up and replaces the previously installed configuration file. The silabs-border-router package first appends the extension ‘.old’ to any existing configuration files to be overwritten. Next, a corresponding file with the extension ‘.new’ is copied from the SILABS_BORDER_ROUTER_ROOT directory to its destination, with the ‘.new’ extension removed. This behavior is equivalent to the following commands:

mv /etc/network/interfaces /etc/network/interfaces.old
cp /opt/siliconlabs/threadborderrouter/interfaces.new /etc/network/interfaces
All instances where major configuration modifications are required are considered ‘Critical’ for the default operation of the Border Router. Customizing these files is not recommended. Because they have nontrivial interactions with other core systems, a modification to one critical file usually requires modifications to many other subsystems within the operating system. The following table describes the major configuration files overwritten by the silabs-border-router package.

### Table 4.4. Major Configuration Files

<table>
<thead>
<tr>
<th>Destination File</th>
<th>Source file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/hosts</td>
<td>hosts.new</td>
<td>Sets the local resolve table used by the local DNS and mDNS services</td>
</tr>
<tr>
<td>/etc/hostname</td>
<td>hostname.new</td>
<td>Sets the hostname the Border Router will use for DNS and mDNS services</td>
</tr>
<tr>
<td>/etc/network/interfaces</td>
<td>interfaces.new</td>
<td>Configures the Border Router’s network interfaces to read border-router.conf</td>
</tr>
<tr>
<td>/etc/init.d/isc-dhcp-server</td>
<td>isc-dhcp-server.new</td>
<td>Forces the DHCP server to start in IPv4 and IPv6 mode (default is IPv4-only)</td>
</tr>
<tr>
<td>/etc/init.d/tayga</td>
<td>tayga.new</td>
<td>When enabled, causes the NAT64 transition technology service to run, allowing Thread devices to target IPv4-only servers</td>
</tr>
<tr>
<td>/etc/default/tayga</td>
<td>default-tayga.new</td>
<td>Overwrites defaults to ensure proper operation of NAT64</td>
</tr>
</tbody>
</table>

### 4.2.6 Templatized Configuration Files

In instances where a configuration must be continuously modified in during the Border Router’s operation, the silabs-border-router package uses a template to configure hook scripts that then dynamically overwrite the configuration file. These hook scripts execute when certain network-related events occur, such as obtaining an IPv6 address via Prefix Delegation – and the resultant configuration file is likely to be overwritten. If your application requires a modification to any of these configuration files, you must either modify the hook scripts or the template to ensure your changes persist in configuration. For ease of reference, template files are installed into the same directory as the configuration files they modify and are postfixed with a `.template` extension. When the triggering event that requires a configuration rewrite occurs, these templates are combined with values from border-router.conf before the hook script replaces the previous configuration. The following table describes ‘Templatized Configuration Files’ that are configured by the silabs-border-router package to dynamically overwrite a configuration file during normal operation.

### Table 4.5. Templatized Configuration Files

<table>
<thead>
<tr>
<th>Path</th>
<th>Configuration File Type</th>
<th>Trigger</th>
<th>Rewrite Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/bind/named.conf.options.template</td>
<td>DNS64</td>
<td>MESH_PREFIX or HOST_PREFIX changes</td>
<td>Enables wlan0 and tun0 to use DNS64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>caused by Prefix Delegation</td>
<td></td>
</tr>
<tr>
<td>/etc/dhcp/dhcpd6.conf.template</td>
<td>DHCPv6 server</td>
<td>HOST_PREFIX changes caused by Prefix Delega-</td>
<td>Assigns the new prefix to the wlan0 interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tion</td>
<td></td>
</tr>
<tr>
<td>/etc/ndppd.conf.template</td>
<td>Neighbor Discovery Proto-</td>
<td>MESH_PREFIX changes caused by Prefix Delega-</td>
<td>Proxies NDP for tun0</td>
</tr>
<tr>
<td></td>
<td>col Proxy Daemon</td>
<td>tion</td>
<td></td>
</tr>
<tr>
<td>/etc/radvd.conf.template</td>
<td>Router Advertisement Dae-</td>
<td>HOST_PREFIX changes caused by Prefix Delega-</td>
<td>Advertises the new prefix on wlan0</td>
</tr>
<tr>
<td></td>
<td>mom configuration file</td>
<td>tion</td>
<td></td>
</tr>
<tr>
<td>/etc/tayga.conf.template</td>
<td>NAT64</td>
<td>MESH_PREFIX or HOST_PREFIX changes</td>
<td>Adds the new prefixes to the ‘allow’ list for NAT64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>caused by Prefix Delegation</td>
<td></td>
</tr>
</tbody>
</table>
4.2.7 Default Network Interface Configuration

The Border Router Add-On Kit uses three default network interfaces:

- Ethernet (eth0): Represents the backbone interface
- Wi-Fi (wlan0): Provides a wireless host access point
- An 802.15.4 interface (tun0): Provides access to the Thread mesh network

The network interfaces can be configured through modifying the /etc/siliconlabs/border-router.conf file. Subnet assignments for the silabs-border-router package are:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Type</th>
<th>IPv6 Subnet</th>
<th>Type</th>
<th>IPv4 Subnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wi-Fi (wlan0)</td>
<td>SLAAC</td>
<td>HOST_SUBNET::/64</td>
<td>DHCP</td>
<td>192.168.42.1/24</td>
</tr>
<tr>
<td>Ethernet (eth0)</td>
<td>SLAAC</td>
<td>DOMAIN_SUBNET::/64</td>
<td>DHCP client</td>
<td>Dynamically assigned</td>
</tr>
<tr>
<td>Ethernet (eth0)</td>
<td>DHCP client</td>
<td>Dynamically assigned</td>
<td>DHCP client</td>
<td>Dynamically assigned</td>
</tr>
<tr>
<td>802.15.4 (Thread)</td>
<td>SLAAC</td>
<td>MESH_SUBNET::/64</td>
<td>Not applicable</td>
<td></td>
</tr>
</tbody>
</table>

On installation, the silabs-border-router package configures the wlan0 and Thread subnets to a static documentation-only address. However, if ENABLE_PREFIX_DELEGATION is set in border-router.conf, these addresses are overwritten whenever a real IPv6 address is obtained from a Prefix Delegating external router connected to the Ethernet interface.

**IMPORTANT:** The default 2001:0db8::/32 IPv6 subnets are used for "documentation only" and should not be used on the IPv6 internet.

4.2.8 IPv6 Packet Forwarding and Router Advertisements

The Border Router must forward IPv6 packets between interfaces so that IPv6 devices on different interfaces such as wlan0 and eth0 may communicate with each other. The Border Router must also reject router advertisements for interfaces on which it is a router, such as wlan0, and accept router advertisements for interfaces on which it is not a router, such as eth0. Refer to Table 4.1 Default Border Router Configuration Settings on page 24 through Table 4.5 Templatized Configuration Files on page 26 to identify the components responsible for packet forwarding and router advertisements, and consult their corresponding configuration files for commentary on how the silabs-border-router enables this feature.
4.2.9 The Neighbor Discovery Protocol Proxy Daemon (ndppd)

Neighbor Discovery Protocol is specified in RFC 4861 and is a foundational component of IPv6 reachability, routing, and SLAAC. In order to discover neighbors on an IPv6 network, a router issues ‘Neighbor Solicitations’ designed to discover if a node exists at a target IPv6 address. Neighbor Solicitations used in this fashion are similar to the use of ‘ARP’ in IPv4: a node is expected to reply to any Neighbor Solicitations for its address with a ‘Neighbor Advertisement.’ Because Thread 6LoWPAN only implements a subset of IPv6 in order to save power, mesh devices do not provide Neighbor Discovery Protocol. Instead, the Border Router acts as a proxy for all Neighbor Advertisements directed to its mesh devices. The comments in the ndppd daemon’s configuration files explain in-detail how the silabs-border-router package enables this feature.

4.2.10 mDNS

Avahi is an implementation of the Apple mDNS bonjour protocol, which allows devices to register and advertise a human-readable name with other devices on the local network. The Border Router publishes the Border Router name with the avahi daemon via IPv4 so that the iOS or Android commissioning is aware of the Border Router. The configuration file is /etc/avahi/avahi-daemon.conf and does not require modification. The avahi daemon is managed by the border-router-app service, which is detailed in section 4.2.2 Silicon Labs Host Applications.

4.2.11 NAT64

Network Address Translation IPv6 to IPv4 (NAT64) is a transition technology that enables IPv6-only mesh devices to communicate with IPv4-only destinations by translating packet headers from IPv6 to IPv4 and maintaining a translation table for return communication. In order to provide an IPv6-only list of destinations to the IPv6 side of the NAT, a NAT64 daemon utilizes a /96 subnet, using the 32-bit suffix to map the entire IPv4 address space. When a packet arrives on the 96-bit NAT64_SUBNET, the NAT64 daemon removes the IPv4 destination address from the lower 32-bits and inserts it into the destination field of an IPv4 packet before substituting a ULA4 address as the source. Finally, before egressing at the internet-facing interface, the Border Router performs NAT44 to substitute the private, internal ULA4 source address assigned by the NAT64 daemon for its external address and a return port. The packet, now translated to IPv4, can traverse the NAT64 only network to its destination, which can respond using the Border Router’s source address and port. Because table entries do not appear in NAT64 until an IPv6 device initiates communication, internal Thread devices will not be available to the external network until after they initiate communication. If your application requires Thread devices to be reachable through the NAT at a static destination, NAT44 port mapping must be configured to map an external port to an internal Thread device. See /etc/tayga.conf and the Tayga man page for more information.

4.2.12 DNS64

The Dynamic Name System for IPv6 to IPv4 (DNS64) is a transition technology that provides name resolution services for the IPv4 internet to IPv6-only clients. DNS clients contacting the DNS64 server will obtain a special IPv6 destination address on a /96 prefix with an IPv4 address appended in the lower 32-bits to which they will send their traffic. By paring DNS64 with NAT64, an IPv6-only device can look up destination addresses that reside within the NAT64 translation prefix, allowing communication with IPv4-only destinations. The Border Router Add-On Kit uses BIND9/named as its DNS64 server, see /etc/bind/named.conf.options or the manpage for bind for more information.

4.3 Utilities, Files and Services

The following utilities, files and services may be useful for Border Router configuration and development.

4.3.1 Write/Read a Disk Image to/from the microSD Card

It is possible to write a disk image to the microSD card or read a disk image from the microSD card using a Windows, Mac or Linux computer. The reason to do this would be to copy the operating system disk image to the microSD card for first time installation, or to read the file system from microSD card to archive or duplicate the disk image. From a PC use Etcher, and from Linux/Mac use the dd command.

4.3.2 Expand the microSD Card File System

The file system may be expanded to the full capacity of the microSD card. The reason to do this would be to enable the full microSD card to be used by the Border Router. From the Border Router use the sudo raspi-config command.

4.3.3 Change the Keyboard Settings

The Raspberry Pi uses an English (UK) keyboard mapping by default. Users may need to change this setting in order to access special characters. From the Border Router use the sudo raspi-config command.
4.3.4 Remote Login to the Border Router

It is possible to login remotely to the Border Router once connected to the Border Router Wi-Fi access point or Ethernet. The reason to do this would be to avoid needing a separate monitor and keyboard. From a PC, open an SSH session using PuTTY at 192.168.42.1, port 22. From a Linux/Mac, use the \texttt{ssh} and \texttt{ssh-copy-id} commands to open a session to \texttt{pi@192.168.42.1}. Login with the default username “pi” and password “raspberry.” Note that the SSH server must first be enabled with the \texttt{sudo raspi-config} command.

4.3.5 Transfer Files To and From the Border Router

It is possible to transfer files between a Windows, Mac or Linux computer and the Border Router once connected to the Border Router Wi-Fi access point or Ethernet. The reason to do this would be to copy the Thread stack to the Border Router for the purpose of building applications as described in section 3.2 Build the Border Router Software Components. These instructions use Mount to access a shared drive on another workstation.

Alternatively, from a PC open a session using \texttt{WinSCP} at 192.168.42.1, port 22. From a Linux/Mac, use the \texttt{scp} command to open a session to \texttt{pi@192.168.42.1}. Login with the default username “pi” and password “raspberry.” Use \texttt{$ ip addr} to find the IP address of the border router. Use the \texttt{wlan0} address if the computer is connected to the Wi-Fi access point. Use \texttt{eth0} if computer is not connected to access point.
5. Resources

5.1 Getting started with Thread
http://www.silabs.com/products/wireless/Pages/thread-getting-started.aspx

5.2 Thread Training Center
http://www.silabs.com/products/wireless/Pages/thread-networking-learning-center.aspx

5.3 Documentation
• UG110: Ember® EM35x Development Kit User Guide
• QSG113: Getting Started with Silicon Labs Thread
• UG103.11: Silicon Labs Thread Fundamentals
• UG278: Zigbee Cluster Library over IP User’s Guide
• AN1010: Building a Customized NCP Application

5.4 Community & Support
http://community.silabs.com/
http://www.silabs.com/support