UG103.13: RAIL Fundamentals

The Silicon Labs RAIL (Radio Abstraction Interface Layer) library provides an intuitive, easily-customizable radio interface layer that supports proprietary or standards-based wireless protocols. RAIL is designed to simplify and shorten the development process. Developers no longer have to deal with hundreds of registers across multiple products, but can instead rely on a unified software API. RAIL, delivered through the Silicon Labs Flex SDK (Software Development Kit), also makes applications portable across Silicon Labs wireless products.

Silicon Labs' *Fundamentals* series covers topics that project managers, application designers, and developers should understand before beginning to work on an embedded networking solution using Silicon Labs chips, networking stacks such as EmberZNet PRO or Silicon Labs *Bluetooth*®, and associated development tools. The documents can be used as a starting place for anyone needing an introduction to developing wireless networking applications, or who is new to the Silicon Labs development environment.

KEY POINTS

- RAIL overview
- RAIL Library description
- RAIL feature summary
- RAIL 2.x changes

1. Introduction

Even with the many wireless standards available, proprietary wireless is still the only option to communicate with legacy, proprietary networks. Also, using a wireless standard is always a compromise—a protocol designed for the application can be much better optimized for energy consumption, data throughput, or range.

However, the flexibility of proprietary protocols comes with a price. It is usually more difficult to develop such protocols, and they are usually incompatible with newer infrastructures. Security design is especially difficult and this must be considered when designing a proprietary protocol.

To develop a proprietary protocol, you need direct control over the radio hardware. However, a general purpose, multiprotocol capable radio, like the Wireless Gecko (EFR32™) is very complex, and would take months to understand. To shorten this process, Silicon Labs provides:

- Radio Configurator, a tool that can generate a radio configuration from a few input parameters like frequency and bitrate.
- RAIL (Radio Abstraction Interface Layer), a C library, which provides a much simpler interface to control the radio from the application code.

Silicon Labs RAIL is the lowest layer for all networking stacks developed internally by Silicon Labs, as well as by the company's customers and third-party partners. RAIL supports a diverse set of radio configurations and functionality and is one of the key underlying technologies of Silicon Labs wireless products.

To make software portability as simple as possible, RAIL was developed with the following goals:

- RAIL API should be backward compatible within major versions. A code developed for RAIL 2.0 should run with the newest version of the RAIL 2.x release cycle.
- RAIL API should be compatible between supported parts as much as possible. Although there are a few APIs that are not available on all parts, general usage—like receive, transmit and state transitions —should be the same.

2. RAIL Overview

The RAIL library itself is delivered through the Wireless Gecko (EFR32™) SDK, but the supporting tools and example applications are also part of the Flex SDK. In the Flex SDK developers can develop protocols that work directly with RAIL, or configure applications based on the Silicon Labs Connect stack. The Silicon Lab Connect stack provides a fully-featured, easily-customizable wireless networking solution optimized for devices that require low power consumption and are used in a simple network topology. For more information, see *UG103.12: Silicon Labs Connect Fundamentals*.

RAIL components in the Flex SDK are:

- The RAIL library: Provides a programming interface to radio functionality, as shown in the following figure.
- The Radio Configurator: Part of Simplicity Studio, a calculator and interface that allows developers to configure static parameters of the radio physical layer. For details see *AN971: EFR32 Radio Configurator Guide.*
- RAILtest, a sample application that includes a serial command for each RAIL library feature, to allow scripted testing and ad hoc experimentation. RAILtest can be built with any PHY, including 802.15.4 and Bluetooth Smart. Many of the RAILtest serial commands can be used for lab evaluation. RAILTest is also a good starting point for testing various RAIL features and can be used as a reference implementation of many APIs.
- Other example applications: Can be used as is for evaluation and also serve as a starting point for application development.
- Documentation, delivered through Simplicity Studio.

For more information about using the example applications in the Flex SDK to begin development with RAIL, see *QSG138: Getting Started with the Silicon Labs Flex Software Development Kit for the Wireless Gecko (EFR32™) Portfolio*.

Although the RAIL library supports the complete EFR32 portfolio, the radio configurator and the Flex SDK examples are only available on EFR32FG, EFR32MG1x and EFR32BG1x.

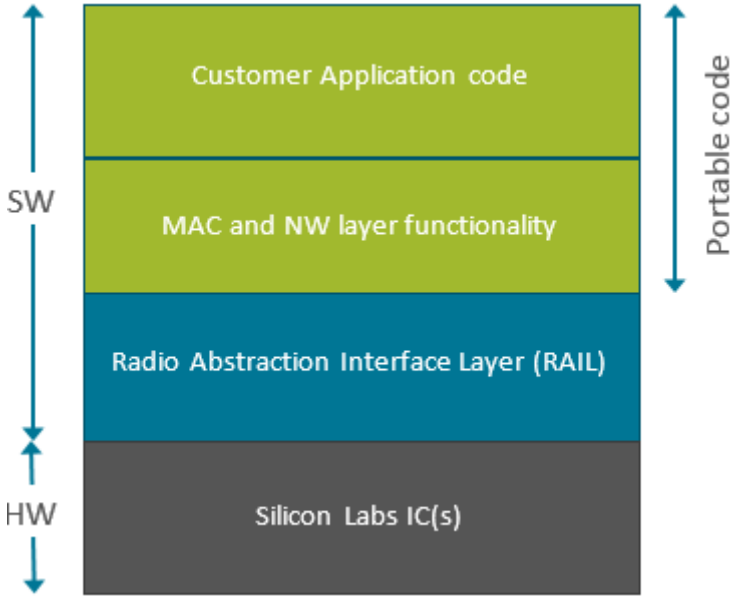


Figure 2.1. RAIL-based Stack Structure

3. RAIL Library

RAIL functionality is delivered as a library that is linked to the developer's application. The RAIL library implements the core features and runtime APIs needed to configure and control the radio.

The RAIL library works by taking intuitive and easy-to-use commands from the RAIL API and translating them into register-level code used to control radio and communications functions. The API commands remain constant across ICs. The changes in the underlying code are transparent to the developer or system tester. This also allows developers to create multiple stacks for different products quickly, as they are always presented with a similar software radio interface. RAIL provides the foundation upon which developers can implement their own MAC layer and network layer functionality.

Where possible, all features currently implemented for the EFR32 will be implemented for future ICs, allowing for easy migration of all RAIL-based applications.

The RAIL API is built up from commands and events. Commands can be used to initiate something from the software, while events are generated by the hardware (usually from an interrupt) to let the software know of something, like a received frame.

RAIL is built to support multiple protocol stacks at the same time. The Silicon Labs Dynamic Multiprotocol solution is implemented with the Radio Scheduler, a preemptive, priority scheduler, which can schedule the radio hardware between protocols. The API is the same for single-protocol and multiprotocol RAIL, but multiprotocol RAIL applications should be implemented somewhat differently. See *UG305: Dynamic Multiprotocol User's Guide* for more information.

The library supports GCC and IAR compilers for ARM.

4. RAIL Features

RAIL includes the following features in software, as well as many others.

RAIL Highlights	Description
Simple Transmit and Receive	Both transmit and receive can be started with a single API. Finished transmission and received packets are signaled with RAIL events.
Automatic State Transitions	Configurable transitions to go from one state to the next based on successful or unsuccessful transmit and receive operations. These support an optional delay as well to allow for consistent timings from one state to the next.
Frame Buffering	Two configurable size frame buffers—one for receive and one for transmit— are available. Both can hold multiple frames.
Timekeeping, Timestamping, and Timers	RAIL automatically maintains a RAIL timebase with µsec granularity. It can be used for: • Timestamping packets (enabled by default) at different packet positions • Scheduling receive and transmit operations • One general purpose timer with optional timer virtualization in RAIL (called MultiTimer) • Optional synchronization with RTC, to keep time and make scheduling possible from EM2 sleep mode
Scheduled Transmit	Transmit at an absolute or relative time in the future.
Scheduled Receive	Start receiving at an absolute or relative time in the future with an optional timeout after that start time.
CCA (Clear Channel Assessment) with Retransmission	Supports two common medium access methodologies, which delays transmission until the channel is clear: • CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) -- based on IEEE 802.15.4 specification. • LBT (Listen Before Talk) -- based on ETSI EN 300 220-1 specification.
RSSI (Received Signal Strength Indicator) Read	Ability to read instantaneous and average RSSI values from hardware. RSSI and Link Quality Indicator (LQI) measurements for each received frame is also available.
Address Filtering	• Two address fields are supported (with fixed location), both maximum four bytes long. • Each field can filter for four addresses (for example, unicast and broadcast address). Addresses can be enabled and configured at runtime. For IEEE 802.15.4, this is handled specially, since it uses addresses with the location configured with the frame type.
Auto ACK	Automatic transmission of a response packet after some delay to every successful receive. This can be a constant payload or dynamically loaded by the user any time before the transmission is supposed to begin. For IEEE 802.15.4 this is handled automatically to improve performance.
IEEE 802.15.4 Helper Features	These features include: • Address filtering (supporting three short address, long address and PAN ID) • Auto ACK • Event on Data Request command to set the frame pending bit **Note:** To implement IEEE 802.15.4-based protocols, Silicon Labs recommends using the Connect stack's MAC mode, which is a full IEEE 802.15.4 MAC layer implementation. For more information, see *UG235.03: Architecture of the Silicon Labs Connect Stack*.
Bluetooth LE and Z-Wave Helper APIs	Helpers for implementing a Bluetooth LE or Z-Wave networking stack. These provide special modes and optimizations in hardware for these specific modes.
Multi-PHY Support	The PHY configuration can be changed runtime. Also, RAIL can load PHY.

RAIL Highlights	Description
Multiprotocol Support	Separate version of the RAIL library with a radio scheduler that can manage requests from different radio configurations on the same chip. • Supports time-slicing N different radio configurations where N >= 2. • Allows for different priority tasks from the different configurations and manages them so the highest priority is always completed. • Allows for tasks to be moved around if the timing isn't critical, for most robust network.
Calibration Support	Chip-specific calibrations. Commonly these include recalibrating for large temperature changes while in receive and image rejection calibration for increased sensitivity.
Stream Generation for RF Testing	RAIL can generate Continuous Wave and PN9 pseudo-random modulated streams with any PHY for RF testing.
Receive Antenna Diversity	Allows toggling between two antennas in receive mode and selecting the one with the stronger signal to improve sensitivity. Note that this is only supported on certain chips and PHY configurations.
RAIL API to Get Entropy from Radio	Radio can be used to collect entropy from receive chain noise.

5. RAIL 2.x Changes

As RAIL has evolved and grown, it became apparent that a significant refactoring of the RAIL code was necessary to achieve the longer term goals of RAIL. As a result, Silicon Labs developed RAIL 2.x.

The most significant changes from RAIL 1.6 to RAIL 2.x are:

- Updated APIs to allow for Dynamic Multiprotocol functionality.
- Unified naming across all APIs to follow a strict VerbNoun naming convention.
- A more powerful channel-based approach to configuring the radio. This adds the ability to automatically change radio configurations and the maximum output power on a per channel basis.
- A unified callback mechanism for all RAIL events.
- A more advanced packet receive API to allow for greater control over when to process incoming packets.

These are the motivations behind RAIL 2.x.

Enable Dynamic Multiprotocol operation:

- Support multiple radio configurations simultaneously on one radio using time slicing.
- Allow multiple instances of RAIL.
- Enable multiple priority levels to support overriding long-running but low-priority radio operations with higher priority radio operations from other RAIL instances.
- Permit each transmit and receive radio operation to specify bounds on when it must execute. If it cannot be scheduled within these bounds, it is canceled and reported as dropped to the calling stack.

Note: RAIL 2.x includes core features for Dynamic Multiprotocol support. Currently, Silicon Labs supports Zigbee-Bluetooth, Bluetooth-RAIL, and RAIL-RAIL DMP applications. Other combinations will be available in the future.

Improve usability and readability of the code:

- Conform to Silicon Labs coding conventions and APIs.
- Streamline callbacks.
- Reduce the number of APIs.
- Simplify the way TX/RX options are specified.
- Simplify Auto-Ack functionality.

Improved functionality and performance:

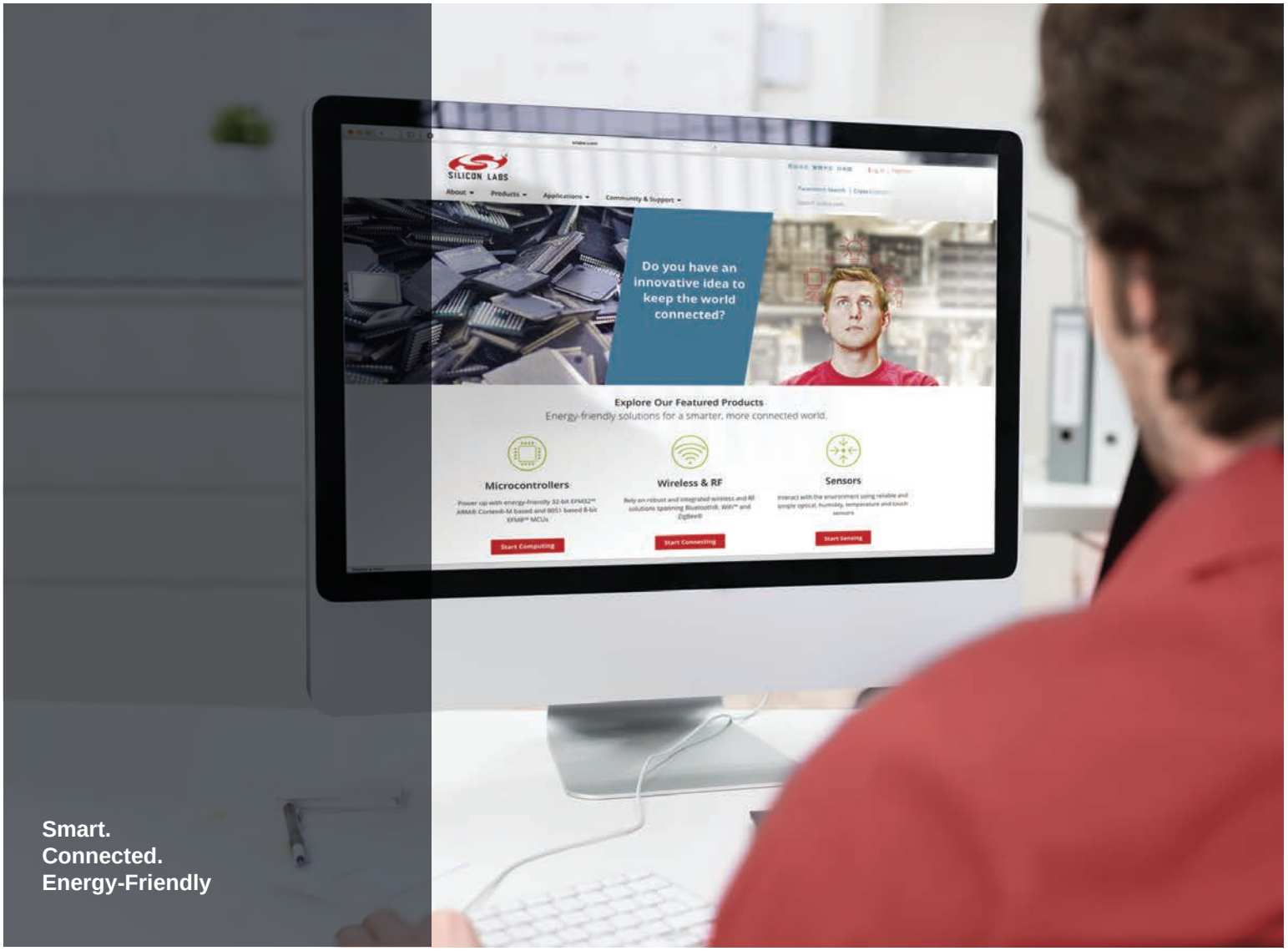
- Power Amplifier (PA) configurability through RAIL.
- Packet Trace Interface (PTI) configurability through RAIL.
- Support for up to three simultaneous IEEE 802.15.4 Private Area Network (PAN) identifiers, short addresses, and long addresses in the filtering logic on the EFR32 series of chips.

While the RAIL 2.x changes are far-reaching, many of them are largely cosmetic, and none of them should remove any previous functionality. See *AN1113: Porting RAIL Applications to RAIL Version 2.x* for more information.

6. Next Steps

See *QSG138: Getting Started with the Silicon Labs Flex Software Development Kit for the Wireless Gecko (EFR32™) Portfolio* for instructions on how to install Simplicity Studio and get started using RAIL from the Flex SDK.

To get more familiar with software development in RAIL, see the API documentation and the Silicon Labs RAIL tutorials.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com