DSR

DOING SOFTWARE RIGHT

ZBOSS Datasheet

Version 1.12

1536 Cole Blvd, Suite 325 Golden, CO 80401, USA
www.dsr-iot.com

Contents

Architecture

ZBOSS Stack Key Architecture Points

Cross-platform	uses OS and hardware-dependent layer for minimum effort porting
OS-less configuration	ability to run without OS support
Multi-tasking	uses internal cooperative multitasking
Application integration	applications are in the same address space as the stack kernel and use the same multi-tasking model
Fixed memory footprint	does not use dynamic memory allocation, which leads to predictable memory budgeting
Optimized memory usage	low RAM capacity on target device – special technique in handling data structures
Zero-copy approach	unnecessary data copy operations are excluded to support more efficient usage of the CPU
Shared memory usage	use of 'packet buffer' data structure for data exchange and memory allocation
Custom build procedure	stack configuration at compilation time, using C preprocessor
Easy-to-use API	inter-layers API uses primitives similar to the ones defined in the Zigbee® and 802.15 standards
Efficient use in Linux	the stack is implemented in Linux as user-space process with applications inside the stack – stack kernel is implemented as a library (library's set)
Optimized power consumption	ZBOSS interrupt-driven I/O to improve battery consumption and exclude polling

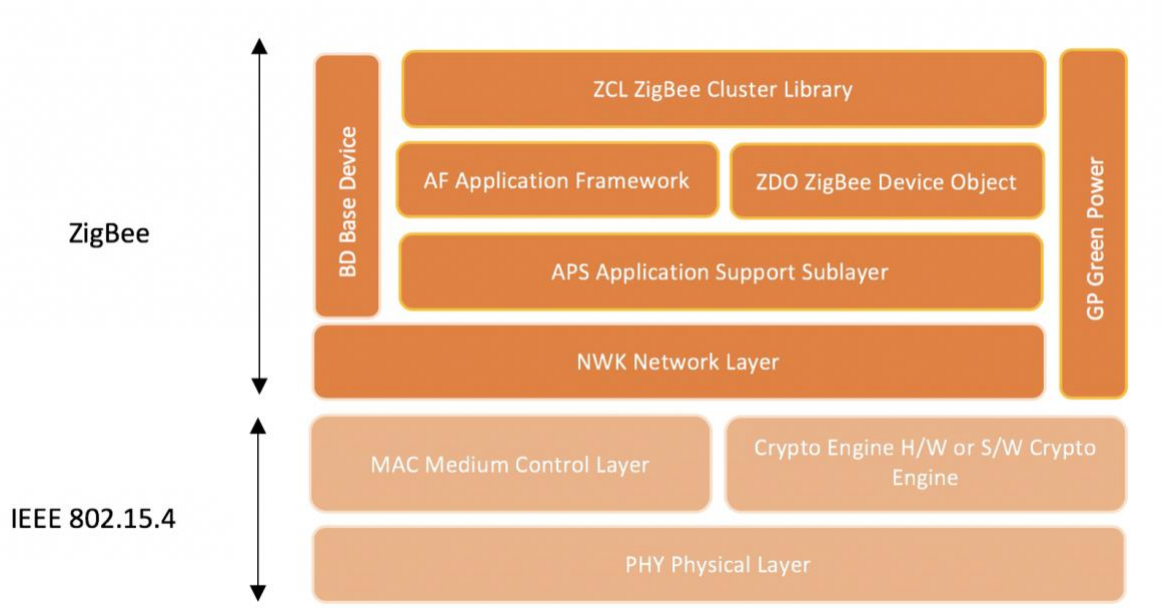
Cross-Platform Support

Cross-platform support is one of the unique features of ZBOSS. To achieve portability, the following concepts were used:

- Cross-platform compatibility was a goal at the start of development and drove the architectural design and decisions
- Hardware-dependent code is isolated in the Hardware Abstraction Layer (HAL)
- Platform-specific logic is localized using macros
- The same stack architecture is used for every platform
- All the implementation logic is in a user-space. Only transport related code runs in a kernel-space (transceiver interruption handling and SPI transport in Linux OS)
- Use of standard solutions when possible (spidev driver in Linux)

Common Architecture

The diagram shows existing ZBOSS layers. Note: currently ZBOSS SDK with SE profile support and ZB3.x version are provided as separate SDKs.



ZBOSS Architecture Options

ZBOSS SDK provides 3 major options to run the stack and an application on top of it:

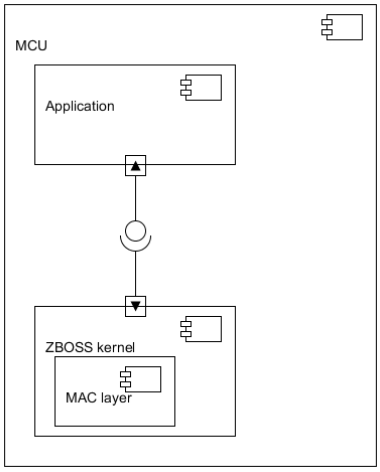
1. Monolithic architecture: both Zigbee stack and application run on the same SoC or MCU
2. Network Co-processor (NCP) architecture: Zigbee stack runs on a SoC, connected to a Host MCU. Application part runs on the host side.
3. Split over MAC architecture. 802.15.4 MAC runs on a SoC, connected to a Host MCU. Zigbee stack and application run on the host.
4. Multi-PAN architecture. A single 802.15.4 radio is used to run two stacks, Zigbee and Thread, simultaneously.

ZBOSS Monolithic Architecture

With monolithic architecture, both the Zigbee stack and the application run on the same SoC or MCU. This architecture is used for products which use a SoC powerful enough to run a full-featured Zigbee stack and application. A Linux platform with a connected radio transceiver is another example where monolithic architecture is used in ZBOSS.

ZBOSS SDK provides:

- Zigbee stack (ZB PRO Core) in binaries
- Zigbee application level (ZCL, BDB, SE1.4) in source files
- Full-fledged ZBOSS API for application development

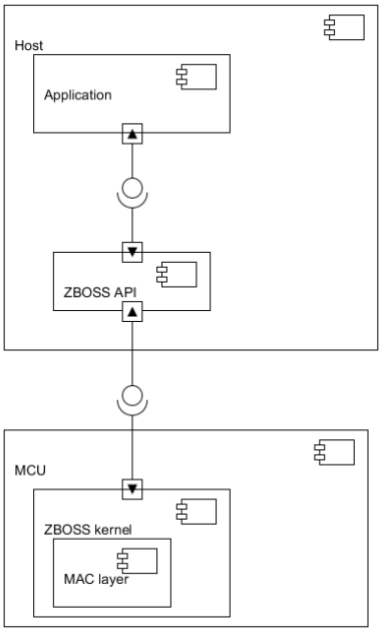


ZBOSS Network Co-processor (NCP) Architecture

Network Co-processor (NCP) architecture is used when a Zigbee SoC is powerful enough to run a full Zigbee stack instance, but there is a need to run an application part on the host to which the SoC is connected via serial transport – UART, USB, SPI, etc. In this case the SoC implements a Zigbee Compliant Platform while the application component runs on the host.
NCP architecture might be a good choice when an isolation of networking and application layers is required, or SoC is not powerful enough to run a monolithic application.

ZBOSS SDK provides:

- Zigbee stack (ZB PRO Core) for SoC in binaries
- Zigbee application level (ZCL, BDB, SE1.4) and serial transport in source files
- Full-fledged ZBOSS API for application development (the same as monolithic SDK API)
- NCP serial protocol, provides low-level ZB networking API (no application level API is provided in this case).

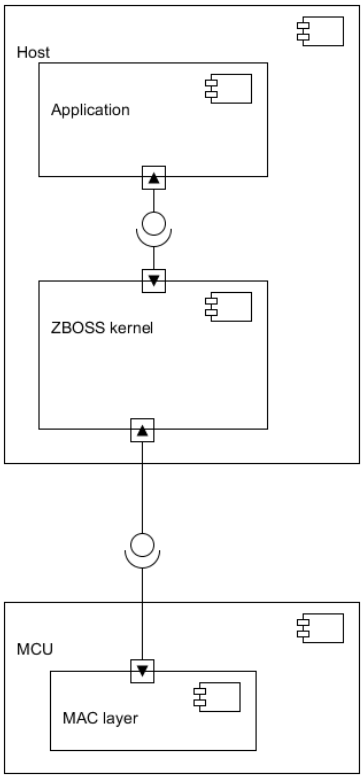


ZBOSS Split over MAC

Split over MAC is most useful when connecting a low-resource Zigbee SoC to a Linux (or other) host. In this architecture, an application and part of the ZBOSS stack run on the host, while ZBOSS MAC runs on the SoC. Serial connection (UART, SPI, USB) is utilized with a minimal speed of 115 kbps.

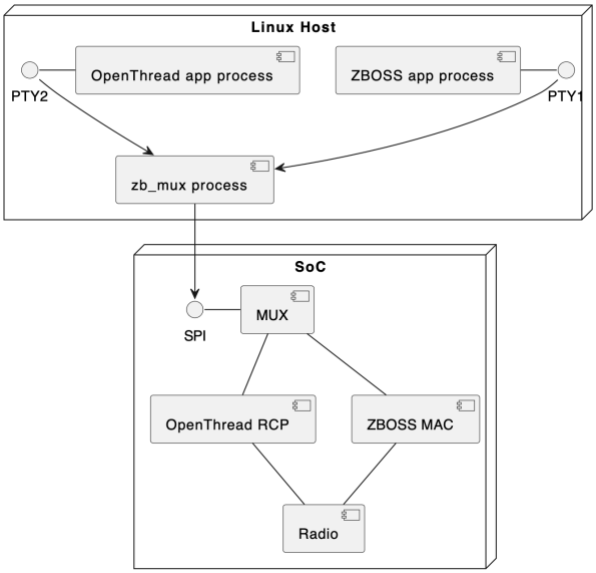
ZBOSS SDK provides:

- Zigbee MAC layer for SoC in binaries
- Zigbee networking layers for a host in binaries
- Zigbee application level (ZCL, BDB, SE1.4) in source files
- Full-fledged ZBOSS API for application development (the same as monolithic SDK API)



ZBOSS Multi-PAN

ZBOSS Multi-PAN architecture allows to run two stacks simultaneously, ZBOSS and OpenThread, using a single 802.15.4 radio. ZBOSS MAC and OpenThread RCP run on a SoC, which is connected to a Linux host using serial connection (UART, SPI). ZBOSS, OpenThread and applications for both stacks run on the host.



OS/Hardware Abstraction

OS/HW platform abstraction is necessary to achieve cross-platform support. C preprocessor is used to support the abstraction layer. The main idea is to avoid using a large amount of “ifdef” pre-processor directives related to portability in the source code, in general, and decrease the number of “ifdef” constructions in the header files not related to the OS abstraction layer. Platform abstraction are implemented as C functions that are placed into OS abstraction layers, while platform-dependent global type declarations and definitions are placed into specific header files.

Global definitions and type declarations can be used anywhere – that is why in the above architecture picture, the OS abstraction layer is displayed as a global layer.

The following objects are platform-dependent:

- Type definitions for base types (8-bit controller vs 32-bit Linux device)
- Definitions for various compilers (gcc, Keil, IAR, etc.)
- Transceiver I/O (interrupt handling for 8051 or ARM vs. file I/O in Linux)
- Wait for I/O (device sleep for 8051 or ARM, wait in `select()` in Linux)
- Trace I/O (UART for 8051 or ARM, SWD for ARM, file in Linux)
- MAC traffic dump (UART for 8051 or ARM, file in Linux)
- Timer (8051, ARM timer at device, `select()` timeout in Linux)

Note that the MAC layer is logically divided into 3 parts:

- Platform-independent MAC logic and API
- Transceiver-dependent (but platform-independent) part
- Platform-dependent MAC transport

Supported Platforms

Hardware Platforms

ZBOSS runs on various hardware platforms. A hardware platform is defined by an MCU, a transceiver and an OS. Below is a list of supported hardware. A target platform may be defined as almost any combination of MCU and transceiver listed below.

Supported Transceivers

- Qorvo (GreenPeak)
  - GP501, GP710, GP711, GP712
- TI
  - CC2520
- CSEM
  - IcyTRX65
- Microchip
  - ATSAMR21G18A
  - MRF24J40
- Synopsys Radio/RF PHY IP
- Several other chipsets under NDA

Supported MCUs

- ARM MCU Core
  - Cortex M3, Cortex M4, ARM9
- XAP5 Core
- 8051 MCU Core
- ARC EM4/EM6 Synopsys Core

Supported SoC

- Nordic
  - nRF52840, nRF52833, nRF52811

- Qorvo (GreenPeak)
  - QPG6095, GP691, GP565
- TI
  - 2.4 GHz SoC: CC2652, CC2530, CC2531, CC2533, CC2538
  - Multi band: CC1352
  - Sub-GHz SoC: CC1310, CC1312
- ON Semiconductors
  - NCS36510
- SiLabs
  - EFR32MG1, EFR32MG12, EFR32MG13, EFR32MG21
- Rafael Micro
  - RT5
- Espressif Systems
  - ESP32-H2
- Telink
  - TLSR8267, TLSR8269
- UBEC
  - UZ2400, UZ2410
- Several other SoC under NDA

Supported OS

- Linux: If Linux OS is installed, any MCU may be used – it doesn't affect ZBOSS functionality. The installed transceiver is an important piece. The following is a list of proven Linux-based platforms:
  - Raspberry Pi
  - Broadcom MIPS (BMIPS5000)
  - Entropic XI3
  - Intel Galileo
- TI-RTOS
- Zephyr
- OS-less: In this case both MCU and transceiver are important for running ZBOSS.
  - Olimex STM32-E407

- STM32F4DISCOVERY

Simulators

- Network Simulator NSNG
- Network Simulator D-Sim (supports virtual time)

ZBOSS Footprint Sample

ZBOSS's footprint (RAM and ROM consumption) is highly dependent on the selected platform and on the included features. The table below defines values of resources required to run ZBOSS-based applications.

The values in the next tables are for ZBOSS 3.0 SE1.4 version, running on TI CC2652 SoC (ARM Cortex-M4 core).

Features	Flash (ROM)	RAM
SE1.4 coordinator WITH sample app	231 Kb	38 Kb
SE1.4 ED WITH sample app	183 Kb	25 Kb

The values below are for ZBOSS 4.0 Zigbee 3.0 version, running on nRF52840 SoC.

Features	Flash (ROM)	RAM
Zigbee 3.0 coordinator with a sample application	226 Kb	31 Kb
Zigbee 3.0 end device with a sample application	164 Kb	18 Kb
Zigbee Direct library	18 Kb	2 Kb

The values below are for ZBOSS 5.0 Zigbee 4.0 version, running on nRF52840 SoC.

Features	Flash (ROM)	RAM
Zigbee 4.0 coordinator with a sample application	241 Kb	42 Kb
Zigbee 4.0 end device with a sample application	175 Kb	18 Kb

ZBOSS Services and Features

Compliance to Standards

ZBOSS supports Zigbee 3.0 and Zigbee 4.0 specification sets:

- MAC: 802.15.4-2011 and 802.15.4-2015, IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)
- Zigbee Specification, revisions 21, 22, 23 and 23.2
- Annex D of Zigbee r23 Specification (Zigbee Sub-GHz)
- Base Device Behavior Specification version 3.0.1 and 3.1
- Zigbee Cluster Library Specification, revision 8
- Zigbee PRO Green Power feature specification Basic functionality set
- Zigbee Direct Specification

MAC 802.15.4

The IEEE 802.15.4 standard defines the two lower layers: the physical (PHY) layer and the medium access control (MAC) sub-layer. ZBOSS implements a MAC sub-layer that controls access to the radio channel using a CSMA-CA mechanism.
ZBOSS MAC's main features and services include:

- Bands supported: 2.4 GHz, Sub-GHz
- MAC data service
    - Data transferring
    - Channel selection
    - CSMA-CA mechanism
    - Frame filtration
    - Auto-acknowledgements
- MAC management service
    - Starting and maintaining PAN

- Channel scanning
- Association/disassociation
- Beacon generation and notification

MAC Implementations

1. ZBOSS upper levels can run on top of one of the following MAC implementations
   - ZBOSS native MAC compiled to ZBOSS
   - Third-party MAC linked to ZBOSS
   - Third-party MAC on SoC connected via serial using customer-specific protocol
   - ZBOSS split over MAC: ZBOSS native MAC via USB/serial/SPI using proprietary protocol
2. For all cases except #1 ZBOSS has special “adapter” layer which translates underlying MAC API into the native ZBOSS MAC API for communication with the upper ZBOSS layers
3. Adapter layer implements ZBOSS traffic dump debug feature as well

Network Layer

The network layer is required to ensure correct operation of the IEEE 802.15.4-2011 MAC sub-layer and a suitable service interface to the application layer.

ZBOSS Network layer’s main services and features include:

1. NWK Data service
   - Unicast/Broadcast/Multicast Communication
2. NWK Management service and maintenance
   - Establishing a new network
   - Permitting devices to join network
   - Neighbor table support
   - Network discovery
   - Joining a network
   - Leaving a network
   - Detecting and resolving address conflicts
   - Managing a PAN ID Conflict

- Routing
- Network parameters configuration
- Frequency agility
- End device timeout protocol and aging mechanism

3. Works With All Hubs (WWAHu)
   - Parent selection

Application Layer

The Application (APL) Layer consists of the APS sub-layer and the ZDO (Zigbee device object). The application support sub-layer (APS) provides an interface between the network layer (NWK) and the application layer (APL) through a general set of services.

The following services are provided by ZBOSS:

1. APS Data service
   - Supported frames: data, command, and acknowledgement
   - Supported data transfer addressing modes: Group, Unicast with short address, and Unicast with IEEE address
2. APS Management service
   - Binding Primitives
   - Information Base Maintenance
   - Group Management
3. Zigbee Descriptors services
   - Supported descriptors: Node, Node power, Simple
4. Device and Service Discovery Client services
5. Device and Service Discovery Server services
6. Network Management Client services
7. Network Management Server services

Green Power

Energy constrained devices, including battery-less devices, make use of the Zigbee PRO Green Power Basic features to securely join Zigbee PRO networks. Such Green Power Devices (GPD) may harvest different amounts of energy depending on the technology used. The Zigbee standard defines several types of

Green Power Infrastructure devices responsible for GPD commissioning as well as receiving and processing Green Power frames.
ZBOSS supports Zigbee Infrastructure nodes: Green Power Proxy Basic and Green Power Combo Basic. DSR's GreenBOSS product compliments ZBOSS and supports Green Power Device side.

To know GreenBOSS Green Power Device features, refer to the GreenBOSS datasheet.

ZBOSS Zigbee infrastructure features:

1. Support for infrastructure device functionality: Green Power Proxy Basic (GPPB) and Green Power Combo Basic (GPCB) devices
2. Support for direct communication with ZGPD and relaying GP data frames from Proxy Basic to Combo Basic device
3. GPD frame translation at Combo device to Zigbee ZCL
4. Support for all Application IDs and ZGPD IDs
5. Commissioning types:
    - Multi-hop commissioning involving both GPPB and GPCB
    - Proximity commissioning involving GPCB only
    - Auto-commissioning for unidirectional communication
    - Bidirectional commissioning for bidirectional communication
    - Application-assisted commissioning (user application can decide to allow or disallow ZGPD commissioning)
    - Manufacturer-specific commissioning (support manufacturer-specific commissioning frame)
6. Communication with GPD in scope of Green Power Basic: unidirectional
7. Partial bi-directional communication with GPD: non-standard for Green Power Basic feature set bidirectional communication with GPD from Green Power Combo Basic device (but not from Green Power Proxy Basic)
8. Support of Security (level 0b11) for bidirectional case

Security

The Zigbee security architecture includes security mechanisms at two layers of the protocol stack. The NWK and APS layers are responsible for the secure transport for their respective frames.

ZBOSS security features are:

1. Standard security mode level 5

2. Trust Center and device roles support
3. Preconfigured Trust Center link key
4. Joining secured network as a router or end device
5. NWK layer security
    - NWK keys management
6. APS layer security
    - End-to-end application keys
    - APS keys management
    - Trust Center link keys management
7. Support for tunneling
8. Install codes support
9. Distributed security
10. Smart energy security
11. Certificate-Based Key Establishment
12. Dynamic Link Key (r23)
13. Device Interview (r23)
14. APS Frame Counter Synchronization (r23)
15. Secured Channel and PAN Changes (r23)
16. Smart Energy Key Management Improvements (r23)
17. Trust Center Swap Out (r23)
18. Restricted Mode (r23)

ZCL

ZCL is a repository for cluster functionality. ZCL consists of a set of elements (such as frame structures, attribute access commands, and data types) and a number of cluster sets.

ZBOSS ZCL features include:

1. Well-predicted and small memory footprint:
    - No dynamically allocated memory at runtime
    - Static memory model is used
2. Size-effective executable code: easy to exclude unnecessary functionality (clusters, commands, handlers, etc.) from build

3. Support for both ZCL client and server roles
4. Support for ZCL general commands
5. Implementation compliance with ZCL revision 8
6. Manufacturer-specific clusters support

The table below lists supported ZCL clusters.

ZCL Cluster Name:

Alarms		On/Off
Basic		On/Off Switch Config
Binary Input		OTA Upgrade
Color Control		Poll Control
Dehumidification Control		Relative Humidity
Diagnostics		Measurement
Door Lock		Scenes
Electrical Measurement		Shade Configuration
Fan Control		Temperature
Groups		Thermostat
IAS ACE		Thermostat UI
IAS WD		Configuration
IAS Zone		Time
Identity		Window Covering
Daily Schedule		Illuminance Measurement
Level Control		Meter Identification
Occupancy Sensing		Power Configuration
Pressure Measurement		Device Temperature Configuration

Analog Input		Carbon Dioxide Measurement
Analog Value		Multistate Input
PM2.5 Measurement		Multistate Value

The table below lists supported SE1.4 clusters.

SE 1.4 Cluster Name:

Key Establishment	Metering
DRLC	Price
Messaging	Tunneling
Prepayment	Energy Management
Calendar	Device Management
Events	MDU Pairing
Sub-GHz	

Base Device Behavior

Base device behavior specification defines the environment, initialization, commissioning and operating procedures of a base device functioning on the Zigbee-PRO stack to ensure profile interoperability.

ZBOSS BDB implemented features include:

1. Security: centralized and distributed
2. Link keys:
    - The default global TC link keys
    - Unique TC link keys
    - The distributed security global link keys
    - An install code derived preconfigured link keys
    - Dynamic Link Key

   - Partner Link Key
3. Commissioning modes:
   - Network steering
   - Network formation
   - Finding & binding
4. For sleepy ED, dynamic data polling based on the operating state of the node

SE1.4

Smart Energy 1.4 profile defines smart energy devices behavior, security procedures and clusters.

ZBOSS SE1.4 features:

1. Multi band radio support:
   - 2.4 GHz radio
   - Sub-GHz radio
   - Multi-MAC “Selection” devices
2. Commissioning. ZBOSS implements procedures for the following SE commissioning modes:
   - Formation and working as Trust Center (ZC only)
   - Auto-join
   - Rejoin to pre-configured PAN
   - Establishment of link key with Trust Center utilizing CBKE procedure
   - Establishment of partner link keys
   - Service Discovery
   - Rejoin and Recovery
   - Steady state
3. Security
   - Installation codes: 48, 64, 96, or 128 bit
   - Cipher suites: Crypto Suites 1 and 2
4. SE cluster – refer to a section describing clusters

Zigbee Direct

Zigbee Direct allows the sending of Zigbee messages via proxy to a BLE device and vice versa. As a result, a device with a BLE radio is able to communicate with devices on a Zigbee network.
ZBOSS supports the following Zigbee Direct roles and services:

- Zigbee Direct Device (ZDD). ZDD has both an IEEE 802.15.4 radio and a Bluetooth Low Energy radio, running BLE in a Peripheral role
- Zigbee Virtual Device (ZVD). ZVD has at least BLE radio, running BLE in a Central role
- Zigbee Direct Security Service
- Zigbee Direct Commissioning Service
- Zigbee Direct Tunnel Service

Production Configuration

ZBOSS supports production time configuration for end products. The idea is to have a number of parameters which may be configured individually for each produced device, such as following:

- TX power
- Installation code
- Device IEEE address
- Certificate
- Application-level data (attributes values, other)

Large Networks Support

ZBOSS Large Networks (large-net) feature allows to support up to 400 or more devices in a network. The large-net configuration mechanism enables the ZBOSS stack to be tuned for both small-scale networks (tens of devices) and large-scale networks (hundreds of devices), optimizing resource utilization.

Appendix I: ZBOSS Middleware

DSR offers integration of the DSR Smart Gateway [SGW] Middleware. It includes everything needed for complete IoT solutions: Gateway Software capable of running on Linux-based or barebone platforms, Scalable Cloud Backend, Customizable Mobile Applications, Administrative and end-user UI, Analytics, and a Firmware Upgrade Server.

Choosing this solution, developed by an experienced team, helps you to save time and effort on the product development, while creating a production-grade, reliable, and secure product.

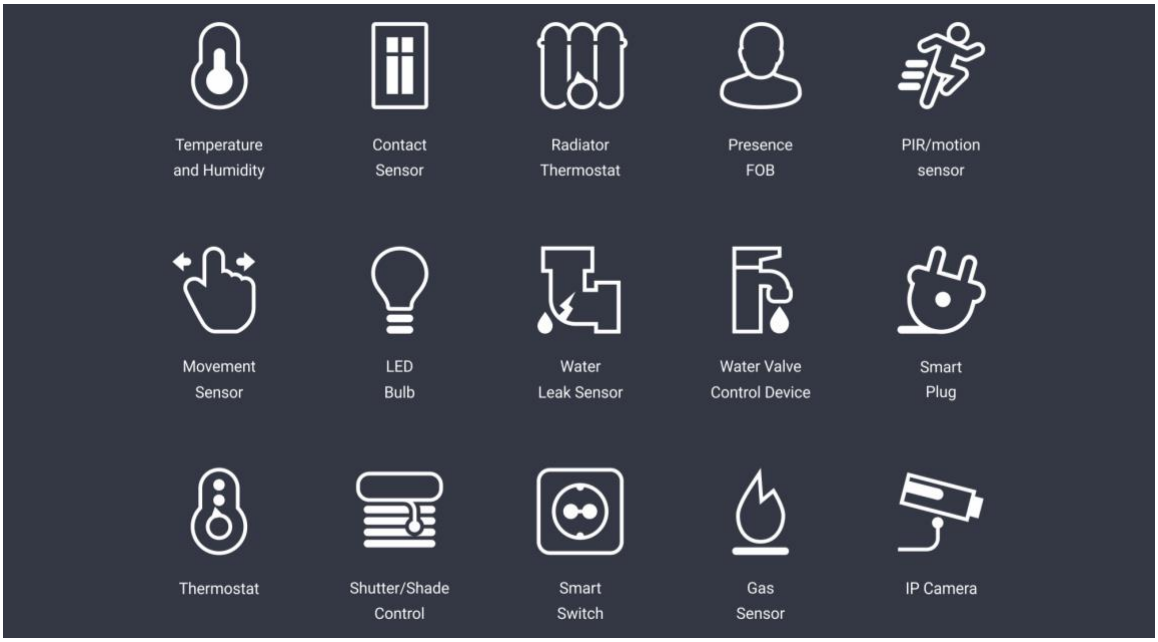
Smart Gateway Functionality and hardware platform(s) can be adjusted according to specific product requirements - from lightweight USB gateways, to set-top-boxes, and Wi-Fi routers.

DSR Smart Gateway is an extendible modular solution allowing for configuration of the software and hardware to fit the customer's specific needs.

Integrated Sensors and Devices

DSR SGW Middleware has integrated support for 180+ devices from more than 40 vendors including Philips, Bosch, Legrand, OSRAM, Bega, IKEA and others.

The following are examples of the device types integrated into DSR SGW Middleware:



Extensions

- Zigbee GW: manage and control ZB network
- Matter Controller: commission and control Matter devices
- BLE GW: manage and control BLE devices
- HTTP API: cloud connection

- FW upgrade service: secure GW upgrade, connected devices OTA FW upgrade
- REX: Rules eXecution engine (local control of scenarios; local data storage)
- Direct API: mobile app direct connection to the GW
- Wi-Fi commissioning: configure Wi-Fi client on the GW
- Self-diagnostic and debug: remote troubleshoot of the GW
- Provisioning and testing: manufacturing time configuration and testing

Reasons to choose Smart Gateway Middleware solution:

- Production ready solution – saves time and development effort.
- Broad ecosystem of connected devices.
- Extendible modular architecture - easy to add new functionality both on hardware/software level (BLE, Wi-Fi, Thread, Matter, etc.)
- Portable solution with support of more than 15 vendors and platforms.
- Extensive support from an experienced team.

Appendix II: DSR SE1.4 Test Harness

The DSR SE1.4 Test Harness (TH) is designed and developed to allow testing on all levels: Application, ZCL, Zigbee PRO networking layer.
The TH key features

- Developed with a rich experience in testing and understanding of necessary verification levels.
- Can be used in automated testing environments.
- Provides extensive test reports required by Test Houses for validation.

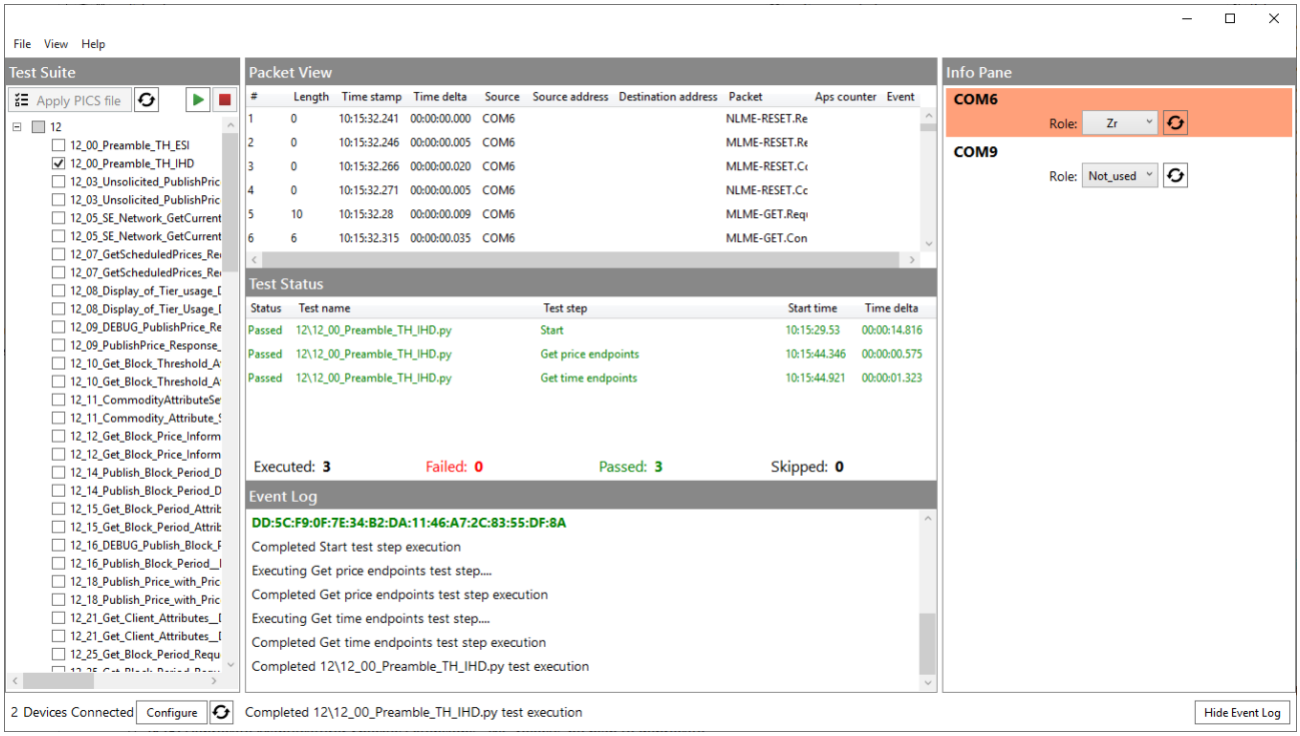
DSR SE1.4 TH is powered by DSR's ZBOSS stack and uses TI CC1352R1 HW as 802.15.4 radio, both 2.4 GHz and Sub-GHz. Its major components are:

- TH engine
- TH UI
- Test scripts library
- TH runs on Linux and Window OS 7 or higher (note: UI is provided for Windows platform only)

SE1.4 TH features:

- **Python language:** TH engine allows for running and validating test scripts written in Python
- **Deep analysis:** the engine allows for deep analysis and verification of packets received, each layer of Zigbee packet and its content may be validated
- **Negative behavior:** the engine is capable of simulating negative behavior as defined in test specifications
- **Automation:** the engine provides an API for running tests in batch mode, allowing for test automation
- **HW independent:** the engine is flexible for swapping 802.15.4 HW. CC1352 is currently used, but it may be replaced with other 802.15.4 HW
- **Tests development:** SE1.4 TH allows custom tests development
- **Ready building blocks:** The rich extension library comes in a package. It contains building blocks for test implementation: widely used Zigbee operations are implemented as ready-to-use functions
- Full SE1.4 certification test set is ready.
- The TH engine is developed to be generic, suitable for testing all existing Zigbee profiles – tests for ZB3.0, ZCL, BDB, GP, etc. can be implemented.

SE1.4 TH User interface:



For questions or additional information, please contact DSR Corporation at contact@dsr-corporation.com.