



Smart home and building solutions.  
Global. Secure. Connected.

# KNX IoT

## Basic Concepts for Device Developers

Hands-on Workshop Zug  
21–22 October 2026



1 Big Picture: KNX IoT, the Point API stack and the two roles · horizontal vs. vertical

---

2 Data Model: functional blocks, datapoints and resources

---

3 Commissioning & Security: discovery, PASE, configuration download, access control

---

4 **Group Communication: Group Addresses link datapoints across devices · multicast vs unicast**

---

5 From Concept to Code: the open-source stack and what to expect in the hands-on

---

# Session Overview

After 45 minutes you understand ...



## The device model

How functional blocks, datapoints and DPTs map to CoAP resources



## Commissioning

How a device gets its Individual Address, keys and configuration (PASE, Load State Machine)



## Group Object linking

How a Group Address links datapoints of different devices via the Function Point Tables



## Message patterns

When to use multicast, unicast or endpoint subscription

**Concepts only.** Hands-on with real devices, the Python client and ETS follows in the SDK and hands-on sessions.



[buildwithknxiot.knx.org](https://buildwithknxiot.knx.org)

[GitLab: knx-iot-point-api](https://gitlab.com/knx-iot-point-api)

# Session Overview

Your map: documentation and code



## **buildwithknxiot.knx.org**

Concepts, checklists, how-tos

- [What is KNX IoT](#)
- [Point API introduction](#)
- [Device Profile \(mandatory checklist\)](#)
- [Device Bootstrapping](#)
- [Function Point Tables](#)
- [Device Access Control](#)
- [Group Communication](#)
- [Stack Features and Versions](#)



## **gitlab.knx.org > knx-iot-point-api**

Open-source code and tools

### **Stack**

C library, Apache-2.0

### **Demos**

LSAB, LSSB, EITT, HEMS

### **Configuration Client**

Python MaC for scripting

### **Schema**

OpenAPI (OAS) of the Point API

### **SPAKE2+ Visualizer**

Step-by-step PASE walkthrough

*Tip: every slide links to the matching page – use the deck as an index into the docs.*

# Big Picture

KNX IoT extends KNX with IPv6-based IETF/IEEE standards – for horizontal interworking between devices and vertical integration into BMS, HEMS or cloud.

# Big Picture

## KNX IoT extends KNX to IPv6 & Unicast/Multicast Group Communication



### Point API

#### Horizontal interworking

Direct IPv6 communication between sensors and actuators; secure S-Mode group communication w/ multicast or unicast; seamless interworking with KNX Classic.



### 3rd Party API

#### Vertical integration

Non-KNX devices and cloud services access a KNX installation via client-server (r/w) and pub/sub.



### Semantic export/import

#### Information model

ETS generates a semantic project export that simplifies integration with non-KNX systems.

**Today's focus:** horizontal interworking via the Point API – what runs inside your device. Vertical integration is out of scope. ETS remains the common tool for KNX Classic and KNX IoT devices.

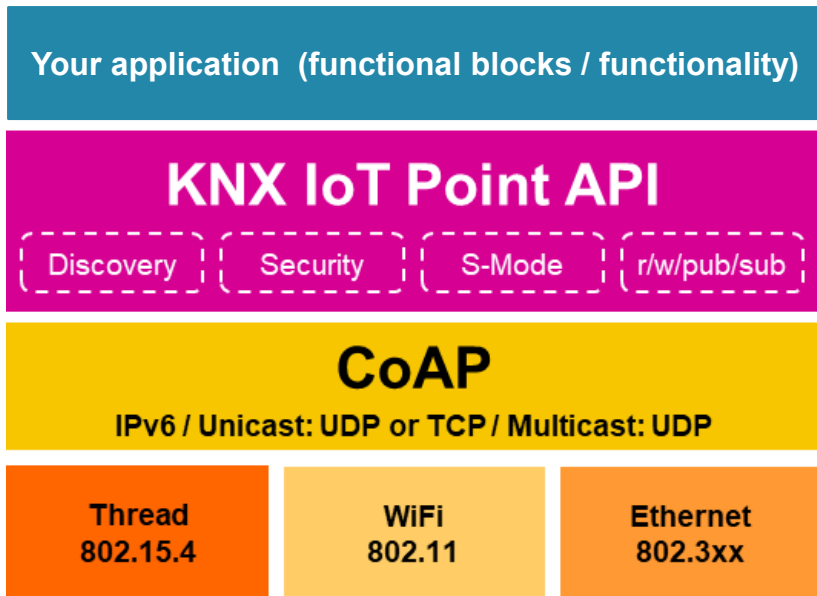


[What is KNX IoT](#)

[Whitepapers \(GitLab\)](#)

# Big Picture

## The Point API stack: standard IETF building blocks



### Point API building blocks



- **Discovery:** DNS-SD / mDNS (RFC 6762) and CoAP /.well-known/core
- **Transport:** CoAP (RFC 7252) over UDP, default port 5683
- **Encoding:** CBOR (RFC 8949), JSON (RFC 8259) optional
- **Security:** OSCORE (RFC 8613) end-to-end; SPAKE2+ (RFC 9383) for onboarding; TLS (RFC 5246) optional
- **Data model:** KNX functional blocks, properties and DPTs – unchanged from KNX Classic



[Point API introduction](#)

[schema.knx.org](https://schema.knx.org)

# Big Picture

## Two roles: Management Client and End Device



### MaC – Management Client

ETS 6 or Python Configuration Client

- Discovers devices
- Authenticates with the device password (PASE)
- Assigns the Individual Address (IA)
- Downloads tables, parameters and keys

*configures*



### KNX IoT End Device

Your product – built on the open-source stack

- Exposes its resources (/dev, /fp, /p, /auth ...)
- Enforces access control on every request
- Sends and receives S-Mode group messages
- Runs the application (e.g. push button, light)

**Scope:** the stack targets the End Device. MaC functionality (a commissioning tool) is not part of the stack.



[Stack Features and Versions](#)

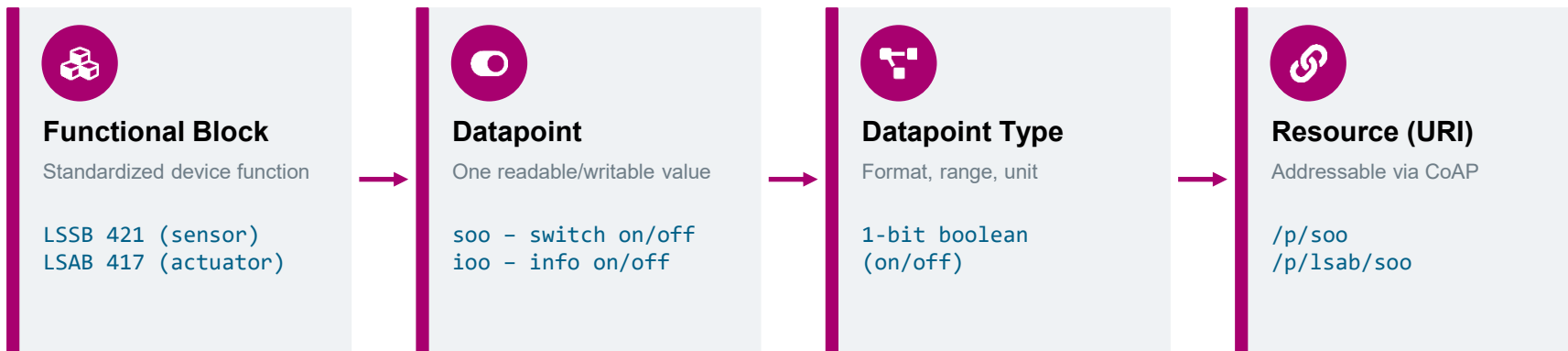
[Python Client Example](#)

# Data Model

The Point API maps KNX functional blocks, properties and datapoint types to RESTful CoAP resources with CBOR/JSON data.

# Data Model

## From functional block to URI



### Example: LSAB datapoint as seen on the wire

```
href : /p/soo
dpa : 417.61 (FB 417, datapoint 61)
if : if.i (input - device consumes)
```

- **if.o** output: device produces and announces the value
- **if.i** input: device receives and consumes the value
- **Same FBs as KNX Classic** → horizontal interworking across manufacturers and media



[ETS Project Example](#)

[Point API introduction](#)

# Data Model

## Anatomy of a device: the resource map



|                        |                                                                                       |                                                 |
|------------------------|---------------------------------------------------------------------------------------|-------------------------------------------------|
| <b>Discovery</b>       | <a href="#">/.well-known/core</a>                                                     | Find devices, resources, group addresses        |
| <b>Device API</b>      | <a href="#">/.well-known/knx</a> · <a href="#">.../ia</a> · <a href="#">.../spake</a> | Reset, IA assignment, PASE handshake            |
| <b>Identity</b>        | <a href="#">/dev/*</a>                                                                | Serial number, IA, programming mode, versions   |
| <b>Application</b>     | <a href="#">/a/lsm</a>                                                                | Load State Machine: unloaded → loading → loaded |
| <b>Function Points</b> | <a href="#">/fp/g</a> · <a href="#">/fp/r</a> · <a href="#">/fp/p</a>                 | Group object, recipient and publisher tables    |
| <b>Datapoints</b>      | <a href="#">/p/*</a> · <a href="#">/f/*</a>                                           | Runtime values, parameters, functional blocks   |
| <b>Security</b>        | <a href="#">/auth/at</a>                                                              | Access tokens: credentials + scope              |
| <b>Group comm.</b>     | <a href="#">/k</a>                                                                    | Receives S-Mode group messages (POST / Observe) |



[Device Profile](#)

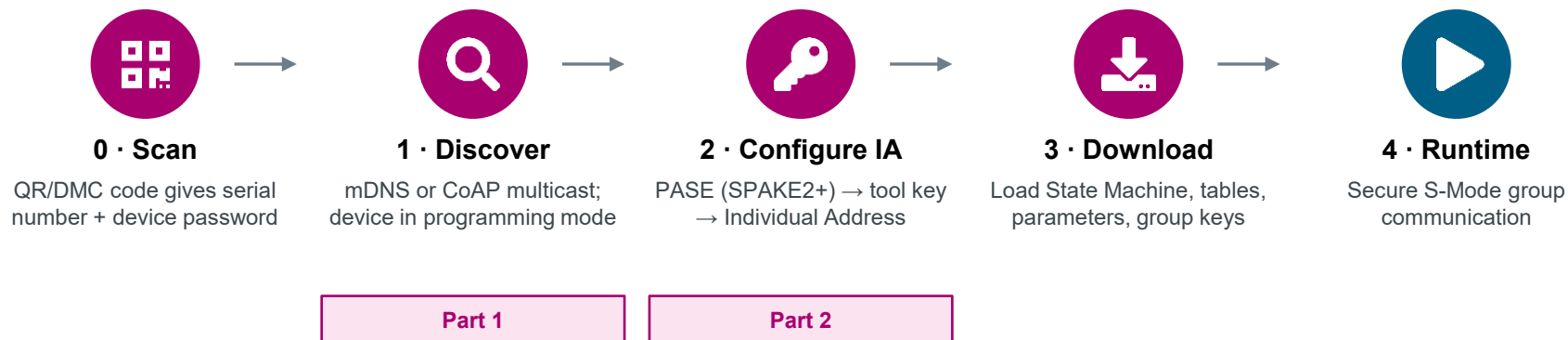
[schema.knx.org](https://schema.knx.org)

# Commissioning & Security

ETS commissions devices via SPAKE2+ authentication; OSCORE then secures group communication and property access.

# Commissioning & Security

## Commissioning in four steps



### Onboarding barcode – ISO 15418 (example)

```
31QMX3+43S00FD10F1F1FE+3ZEUI:0080E126030A980A.P:PASSWORD
└─model      └─serial no.      └─EUI-64      └─password
```

- Both parts are independent and can run separately
- All traffic after PASE is OSCORE-protected
- Sleepy devices wake up faster while in programming mode



[Device Bootstrapping](#)

[ETS Project Example](#)

# Commissioning & Security

## Discovery: find the device first



### DNS-SD / mDNS

Zero-configuration browsing

```
_knx._udp      service type
<serialnumber> always
_ia<iid>-<ia>   after IA assignment
_pm           in programming mode
SRV → CoAP port 5683
AAAA → all IPv6 addresses
```

- **Thread:** Border Router answers as mDNS proxy – device must run an SRP client
- **Sleepy devices:** TXT record SP=<ms>

### CoAP resource discovery

GET /.well-known/core (unicast or multicast)

```
?ep=knx://sn.<sn>    by serial number
?if=urn:knx:if.pm    in programming mode
```

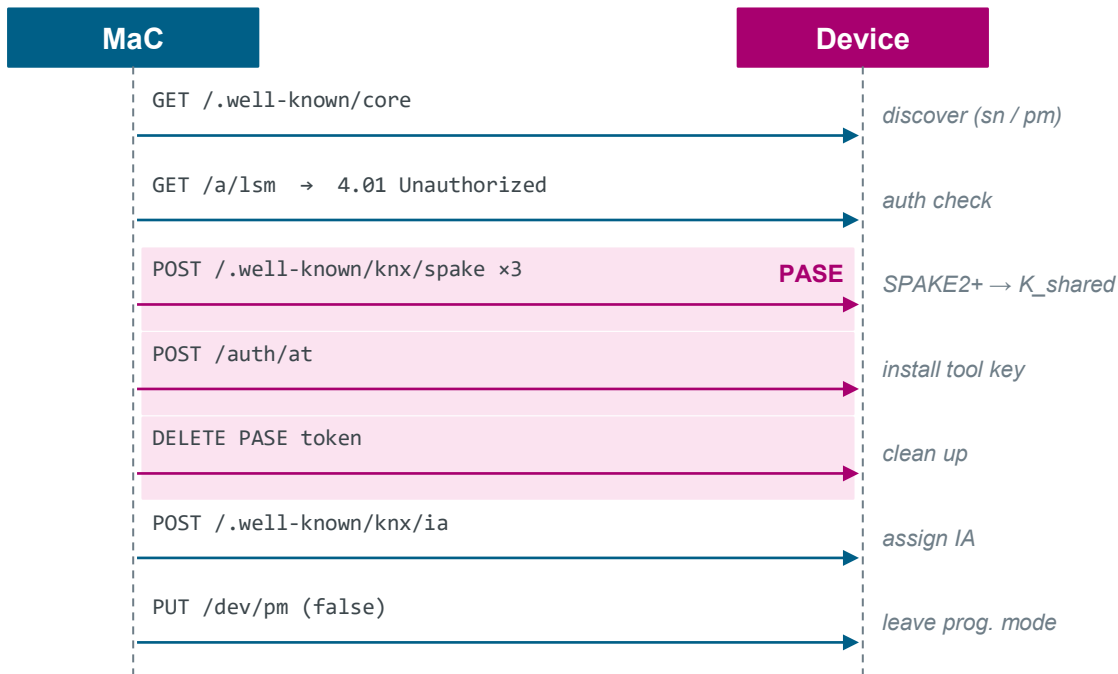
- Only mechanism when no mDNS proxy is available
- **Multicast:** no response on mismatch or error – silence is normal
- Also used at runtime to resolve IA → IPv6 for unicast



[Device Profile](#)

# Commissioning & Security

## Part 1 – PASE: from password to tool key to IA



### Why it matters

- Password never travels over the network
- Mutual authentication of MaC and device
- Short-lived PASE key is replaced by a long-lived OSCORE tool key
- IA identifies the device in the installation from now on

[Try the SPAKE2+ Visualizer →](#)



[Device Bootstrapping](#)

[SPAKE2+ Visualizer](#)

# Commissioning & Security

## Part 2 – Configuration download



/a/lsm



### What the MaC downloads while in loading

|          |                     |
|----------|---------------------|
| /fp/g    | Group Object Table  |
| /fp/r    | Recipient Table     |
| /fp/p    | Publisher Table     |
| /p/*     | Parameter values    |
| /auth/at | Group access tokens |



Device Bootstrapping

Function Point Tables

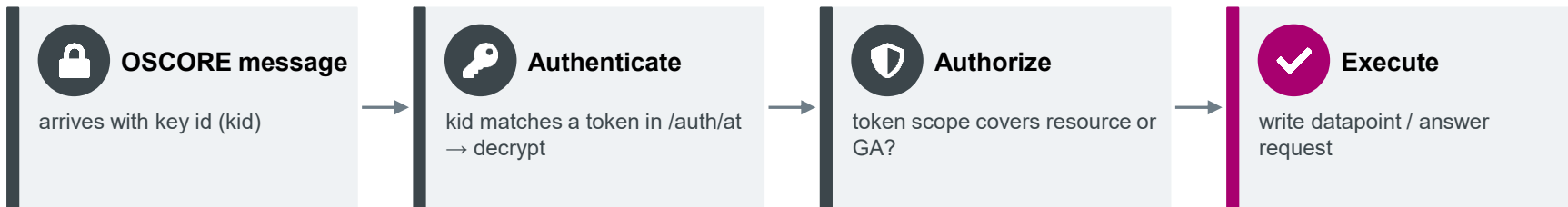


### Rules your device must follow

- **GET** on tables: allowed in every state
- **POST / DELETE** on tables: only in loading
- **Runtime** group communication uses the tables only in loaded
- **Factory reset** via /.well-known/knx – Part 2 starts with a reset that preserves the IA

# Commissioning & Security

## Security model: authenticate, then authorize



### Access token (/auth/at) – key fields

```
id      : "1a"           referenced by at=  
cnf.osc : id (kid), ms (secret)  
scope   : [2304, 2305] | ["if.sec"]  
profile : 2 coap_oscore | 255 coap_pase
```

### Access scopes

|                      |                                   |
|----------------------|-----------------------------------|
| <b>if.i / if.o</b>   | runtime input / output datapoints |
| <b>if.g.s</b>        | all S-Mode group objects          |
| <b>[ga, ...]</b>     | selected group addresses          |
| <b>if.p / if.d</b>   | parameters / diagnostics          |
| <b>if.c / if.sec</b> | configuration / security config   |

*OSCORE contexts are independent of the IPv6 address – a device keeps its secure association after an address change.*



[Device Access Control](#)

[Whitepapers \(GitLab\)](#)

# Group Communication

S-Mode messaging sends (publishes) events from one device to a group of recipients – linked by Group Addresses configured in ETS.

# Group Communication

## S-Mode: event-driven group communication



- **Event-driven:** devices send only on a change or event – less traffic, less energy
- **Producer–consumer:** the sender does not expect a reply; feedback arrives as a separate group event (e.g. status GA)
- **Group Address (GA):** the logical link between datapoints of different devices – configured by ETS
- **Endpoint /k:** group messages are POSTed to /k of the receiver (or to a multicast address)

### Group value message

```
POST coap://[ff35:30:<iid>::<grpid>]/k
OSCORE(kid=...)
{ sia: 5678,
  s: { st: w,
      ga: 1234,
      value: 20.5 } }
```

|              |                                       |
|--------------|---------------------------------------|
| <b>sia</b>   | source Individual Address             |
| <b>st</b>    | service type: write / read / response |
| <b>ga</b>    | group address                         |
| <b>value</b> | datapoint value (CBOR)                |



[Group Communication](#)

[Whitepapers \(GitLab\)](#)

# Group Communication

## Function Point Tables: three tables per device



**/fp/g**

### Group Object Table

Binds a datapoint (href) to one or more GAs + flags. The central table.

**/fp/r**

### Recipient Table

Where outgoing messages are sent /  
Who the recipient is:  
grpid (multicast) or ia + at (unicast).

**/fp/p**

### Publisher Table

Where incoming messages come from /  
Who sends the message:  
grpid (multicast) or ia + at (subscription).

### Group object flags (cflag)



e.g.  $W+U$  = receive and write value ·  $R+T$  = send on change

### Key idea

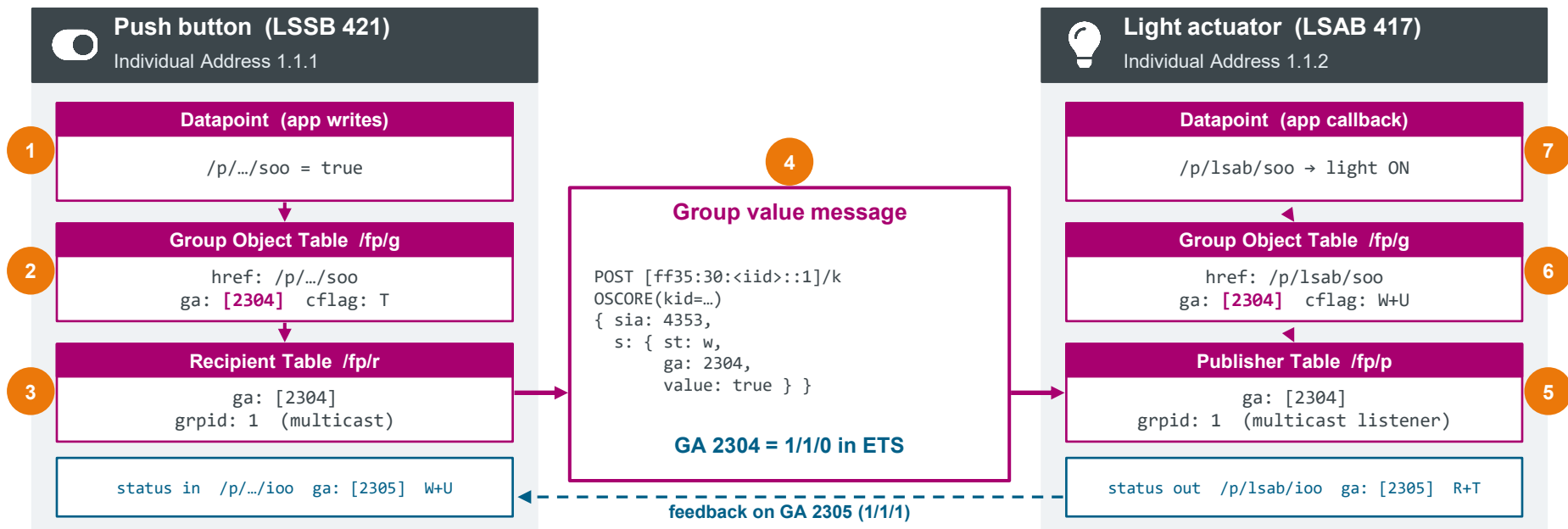
**The GA is the only thing two devices share.** The MaC writes the same GA into /fp/g of each device – that creates the link. /fp/r and /fp/p only decide how the message travels.



Function Point Tables

# Group Communication

Example: one Group Address links two devices



Same GA in /fp/g of both devices = the link. Your app only writes/reads datapoints (1, 7) – the stack does 2–6.



Function Point Tables

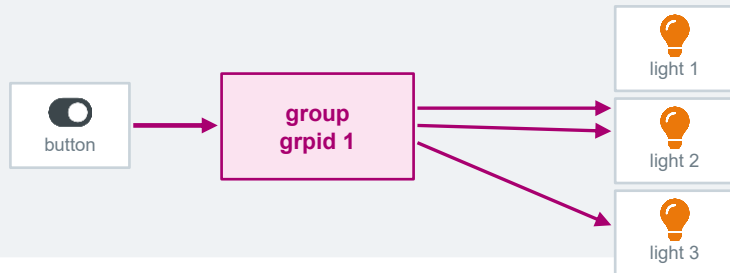
Group Communication

# Group Communication

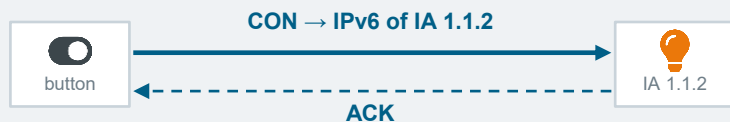
## Same link, two ways to deliver: multicast vs unicast



### Multicast 1 : N · not confirmed



### Unicast 1 : 1 · confirmed



| Changes        | Multicast                    | Unicast                                 |
|----------------|------------------------------|-----------------------------------------|
| Sender /fp/r   | ga + grpId                   | ga + ia + at                            |
| Destination    | IPv6 multicast group         | IPv6 address of one device              |
| Receiver /fp/p | needed (listens to grpId)    | not needed                              |
| Before sending | –                            | resolve IA → IPv6 (mDNS / CoAP), cached |
| Delivery       | not confirmed                | confirmed (CoAP CON)                    |
| Typical use    | all lights in a room, scenes | reliable command, status to one HMI     |

**Stays the same:** GA, /fp/g entry, payload (sia, st, ga, value) and OSCORE protection.

**The GA defines WHAT is linked – the /fp/r entry decides HOW it travels.**



[Group Communication](#)

[Function Point Tables](#)

# Group Communication

## Three message exchange patterns



### Multicast

1 : N · low latency

- One IPv6 multicast message reaches all group members
- Not confirmed
- Sender: /fp/r with grpId · Receiver: /fp/p

**Use for:** Lights in a room, scenes

### Direct Unicast

1 : 1 · confirmed

- CoAP CON to the recipient's IPv6 address
- IA → IPv6 resolved via mDNS/CoAP and cached
- Sender: /fp/r with ia + at · Receiver: no /fp/p needed

**Use for:** HMI status, reliable commands

### Endpoint Subscription

star · CoAP Observe

- Receiver subscribes to /k of the sender
- Sender notifies on change (T flag, GA in token scope)
- Configured in the receiver's /fp/p (ia + at)

**Use for:** Controller collects sensor data, IoT Router (↑ vertical)



[Group Communication](#)

[Whitepapers \(GitLab\)](#)

# From Concept to Code

With the open-source Point API stack and demo apps (LSAB, LSSB), we build, flash and commission devices with ETS.

# From Concept to Code

## Outlook: two ways to act as MaC in the hands-on



### Python Configuration Client

Scriptable, every commissioning step visible

|                              |                                  |
|------------------------------|----------------------------------|
| <code>\$ discover</code>     | Discovery                        |
| <code>\$ configure-ia</code> | Part 1 · PASE, tool key, IA      |
| <code>\$ configure-ap</code> | Part 2 · tables, GAs, parameters |
| <code>\$ smode</code>        | Runtime · send a group message   |

### ETS 6

The tool your integrators will use

|   |                 |                                      |
|---|-----------------|--------------------------------------|
| 1 | <b>Import</b>   | knx_iot_virtual_lsxb.knxproj         |
| 2 | <b>Link GAs</b> | button soo ↔ light soo (GA 1/1/0)    |
| 3 | <b>Download</b> | Topology → Download All              |
| 4 | <b>Test</b>     | Toggle sensor, watch actuator, trace |

**Both do the same:** discover, commission and download the Function Point Tables you saw in block 4.



[Python Client Example](#)

[ETS Project Example](#)

[Configuration Client repo](#)

# From Concept to Code

Next up: from concepts to your own device



## SDK sessions

Port and run the stack on Zephyr, FreeRTOS and silicon-vendor evaluation boards



## Hands-on

Build LSAB / LSSB, link them with a GA, commission them and watch the messages on the wire

*Outlook: vertical integration via 3rd Party integration and the HEMS demo*



## Keep open

buildwithknxiot.knx.org and the developer checklist slide

**Remember:** a Group Address links datapoints of different devices – written by the MaC into the Function Point Tables. Your application only reads and writes datapoints; discovery, security and group addressing are handled by the stack.



[buildwithknxiot.knx.org](https://buildwithknxiot.knx.org)

[GitLab: knx-iot-point-api](https://gitlab.com/knx-iot/point-api)

# Reference

## Cheat sheet



|               |                                              |                      |                                        |
|---------------|----------------------------------------------|----------------------|----------------------------------------|
| <b>MaC</b>    | Management Client (ETS, Python client)       | <b>IA</b>            | Individual Address of a device         |
| <b>GA</b>     | Group Address – logical link for S-Mode      | <b>grpid</b>         | Multicast group ID in /fp/r · /fp/p    |
| <b>FB</b>     | Functional Block, e.g. LSAB 417              | <b>DPT</b>           | Datapoint Type – format, range, unit   |
| <b>dpa</b>    | Datapoint address, e.g. 417.61               | <b>LSM</b>           | Load State Machine (/a/lsm)            |
| <b>/fp/g</b>  | Group Object Table – datapoint ↔ GA          | <b>/fp/r · /fp/p</b> | Recipient / Publisher Table            |
| <b>PASE</b>   | Password Authenticated Session Establishment | <b>SPAKE2+</b>       | Password-based key exchange (RFC 9383) |
| <b>OSCORE</b> | Object security for CoAP (RFC 8613)          | <b>kid</b>           | OSCORE key id → access token           |
| <b>cflag</b>  | Group object flags R W T U                   | <b>S-Mode</b>        | System Mode group communication        |



[buildwithknxiot.knx.org](https://buildwithknxiot.knx.org)

[Whitepapers \(GitLab\)](#)

[schema.knx.org](https://schema.knx.org)

# From Concept to Code

## Developer checklist: what typically trips people up



### Device ignores group messages

LSM not in loaded, or GA missing in /fp/g · /fp/p



### Message silently dropped

No token with matching kid, or GA not in scope (/auth/at)



### Multicast discovery returns nothing

Response suppression on mismatch is required behaviour



### Unicast fails, multicast works

IA → IPv6 resolution needs working mDNS / CoAP discovery



### Thread device not found

SRP client registered with the Border Router? Sleepy: SP set?



### Status lost now and then

Multicast is unconfirmed – use unicast for critical feedback set



[Device Profile](#)

[Device Access Control](#)

[Group Communication](#)

# From Concept to Code

## The open-source stack at a glance



C

portable, event-driven

Apache-2.0

royalty-free

CMake

presets for all targets

### Platforms

Linux

Windows

Zephyr

FreeRTOS

*FreeRTOS: NXP platform; ESP32 planned with stack 1.2.0*

### Two repositories

- [knx-iot-point-api-stack](#) core library + tests
- [knx-iot-point-api-demos](#) LSAB, LSSB, EITT, HEMS – fetches the stack via CMake FetchContent

### Versions (tentative dates)

| Stack             | Point API spec | EITT | ETS   |
|-------------------|----------------|------|-------|
| 1.0.0 (available) | 1.1.0          | V39  | ≥ 6.4 |
| 1.1.0 (06/2026)   | 1.4.0          | V40  | ≥ 6.4 |
| 1.2.0 (12/2026)   | > 1.4.0        | TBD  | TBD   |

### Build it (hands-on)

- **Linux / Windows:** HOW-TO-BUILD.md (presets, MSYS2/MinGW, VS Code)
- **Zephyr:** HOW-TO-BUILD-ZEPHYR.md (workspace, flashing, serial monitor)
- Clone stack and demos side by side to develop against a local stack



[Stack Features and Versions](#)

[How to Build](#)