Messaging across endpoints

The Kaa event subsystem enables messages delivery across the endpoints (EP). Events can be thought of as commands or structured chunks of data. For example, an event from a smartphone application can toggle the lights in the room.

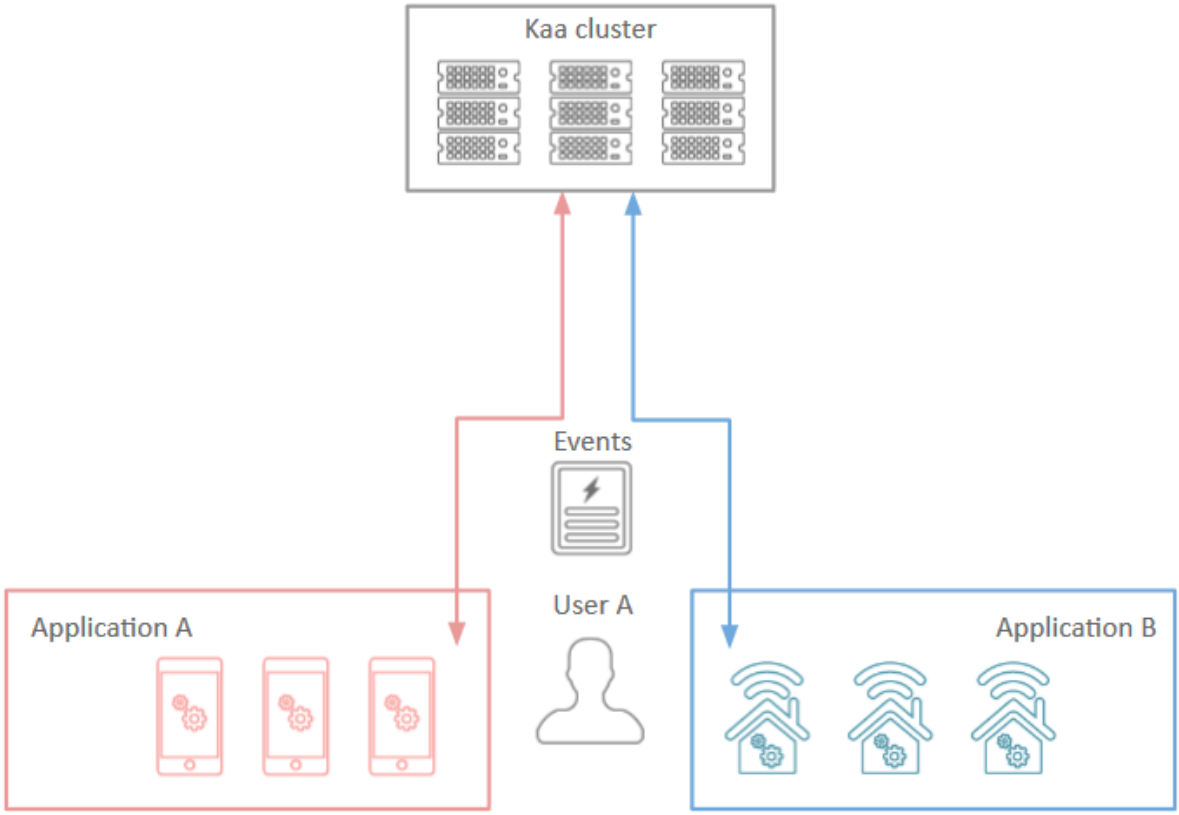
Structure of the data carried by events is defined by an Event Class (EC) data schema configured on the Kaa server and built into Kaa SDK. ECs are grouped into Event Class Families (ECF) by the topic. Kaa allows delivering events among EPs that belong to the same or different applications. As of Kaa r0.6, endpoints must be associated with the same user in order to be able to send events to each other. Please review the events system design reference for more background.

From this guide you will learn how to use Kaa events functionality for enabling communication across the endpoints.

Basic architecture

The following diagram illustrates basic entities and data flows in scope of the event management:

- Events are generated based on the event class schema created by the developer for the event class family
- The event class family can be used by one or multiple applications, thus the event can be shared between applications
- Several applications belonging to the same user share events between each other according to the application mapping and through the Kaa server



Configuring Kaa

This section provides guidance on how to configure ECFs in Kaa.

Creating ECFs

ECFs are shared between applications and there is no default ECFs created automatically. A tenant admin can create multiple ECFs on the Kaa server using the Admin UI or REST API.

In this guide, we will use the following ECF that allows you to remotely control a thermostat using your cell phone.

```
[
    {
        "namespace":"org.kaaproject.kaa.schema.sample.thermo",
        "type":"record",
        "classType":"event",
        "name":"ThermostatInfoRequest",
        "fields":[

        ]
    },
    {
        "namespace":"org.kaaproject.kaa.schema.sample.thermo",
        "type":"record",
        "classType":"object",
        "name":"ThermostatInfo",
        "fields":[
            {
                "name":"currentTemperature",
                "type":"int"
            },
            {
                "name":"targetTemperature",
                "type":"int"
            }
        ]
    },
    {
        "namespace":"org.kaaproject.kaa.schema.sample.thermo",
        "type":"record",
        "classType":"event",
        "name":"ThermostatInfoResponse",
        "fields":[
            {
                "name":"thermostatInfo",
                "type":"org.kaaproject.kaa.schema.sample.thermo.ThermostatInfo"
            }
        ]
    },
    {
        "namespace":"org.kaaproject.kaa.schema.sample.thermo",
        "type":"record",
        "classType":"event",
        "name":"ChangeTemperatureCommand",
        "fields":[
            {
                "name":"temperature",
                "type":"int"
            }
        ]
    }
]
```

Application mapping

Our sample ECF contains three events: `ThermostatInfoRequest`, `ThermostatInfoResponse` and `ChangeTemperatureCommand`. We will create the following two applications in this guide: Controller and Thermostat. It is logical that Controller should be able to send the `ThermostatInfoRequest` event (act as a source) and receive the `ThermostatInfoResponse` event (act as a sink), while Thermostat should be able to send the `ThermostatInfoResponse` event and receive the `ThermostatInfoRequest` event. Based on such logic of our application, we will create the following application mapping for our sample ECF:

Application	ThermostatInfoRequest	ThermostatInfoResponse	ChangeTemperatureCommand
Controller	source	sink	source
Thermostat	sink	source	sink

A tenant admin can set up application mapping using the Admin UI or REST API.

Generating SDK

During the SDK generation, the Control server generates the event object model and extends APIs to support methods for sending events and registering event listeners. An application SDK can support multiple ECFs. However, it cannot simultaneously support multiple versions of the same ECF.

Coding

This section provides code samples which illustrate practical usage of events in Kaa. The event subsystem API varies depending on the target SDK pl atform, but the general approach is the same.

Attach endpoint to user

To enable sending/receiving events to/from endpoints, at first the client should attach the endpoint to the user as shown in the following screenshot.

Assisted attach

Specific endpoint may not be able to attach itself independently. E.g. in case if endpoint doesn't have an user token. Another endpoint that already attached can assist in attachment process of the new endpoint. Below are examples of assisted attachment.

Get ECF factory and create ECF object

To access the Kaa event functionality, the client should implement the two following blocks of code.

Get ECF factory from Kaa

Get specific ECF object from ECF factory

Send events

To send one or more events, the client should proceed as described in this section.

Get endpoint addresses

Execute the asynchronous *findEventListeners* method to request a list of the endpoints supporting all specified EC FQNs (FQN stands for *fully qualified name*).

Send one event to all endpoints

To send an event to all endpoints which were previously located by the *findEventListeners* method, execute the *sendEventToAll* method upon the specific ECF object.

Send one event to one endpoint

To send an event to a single endpoint which was previously located by the *findEventListeners* method, execute the *sendEvent* method upon the specific ECF object and this endpoint.

Send batch of events to endpoint(s)

To send a batch of events at once to a single or all endpoints, execute the following code.

Receive events

To start listening to incoming events, execute the *addListener* method upon the specific ECF object.
