EXOSITE

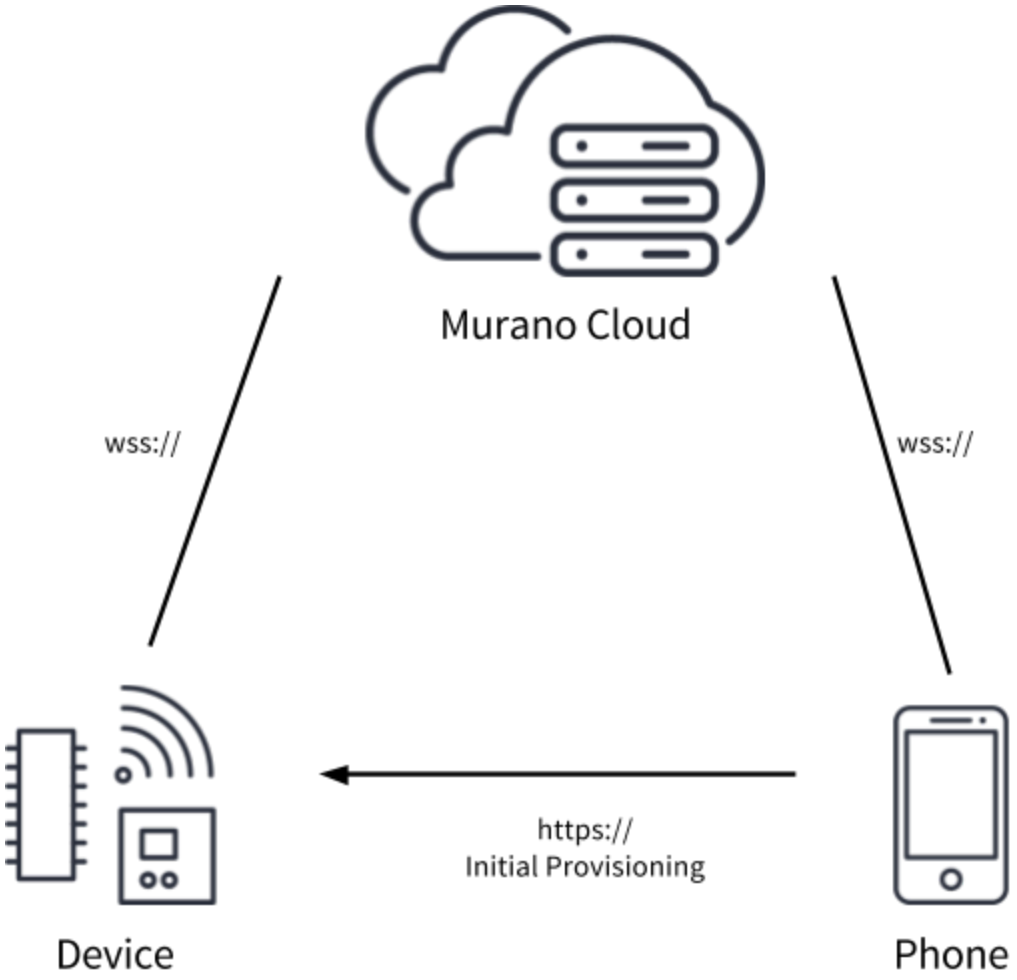
Chungear Demo Application Specification

Data Architecture Specification - DAS

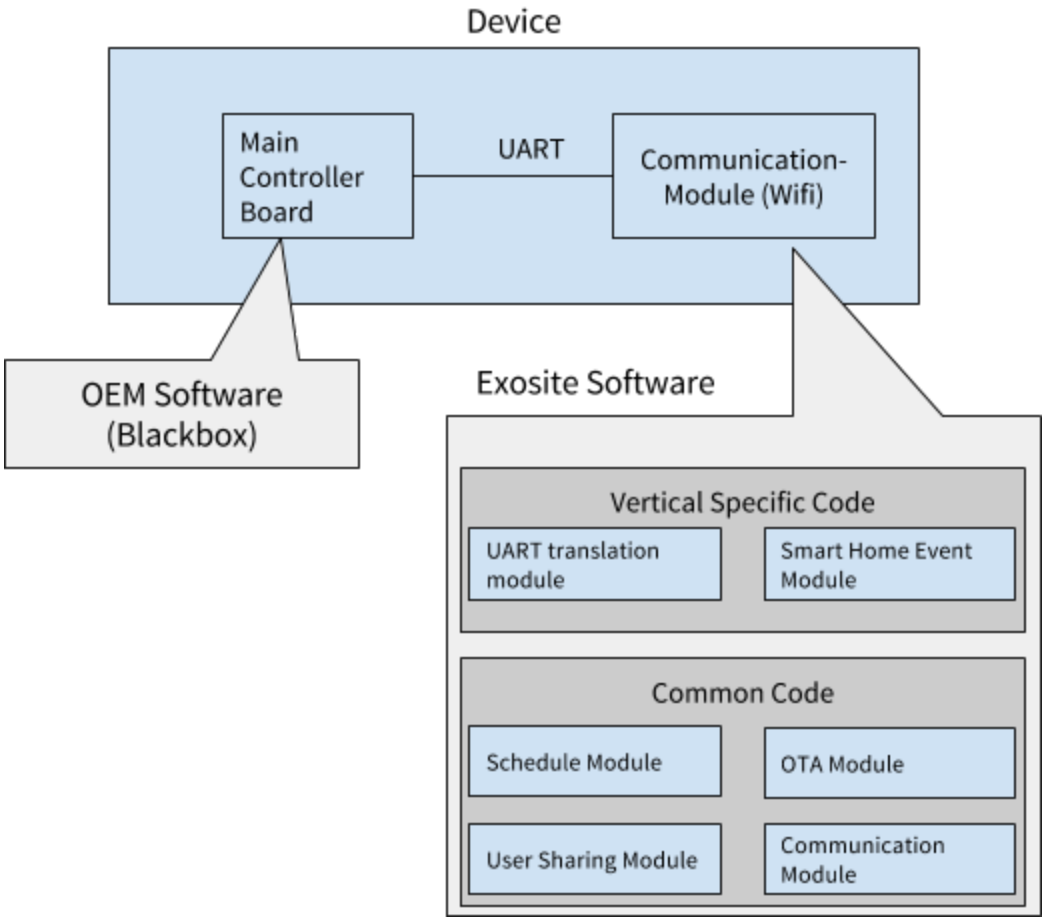
Revision D
2 November 2016

Date	Revision	Description	Author / Editor	Reviewer
14 Oct 2016	A	1st draft	DL	DL
25 Oct 2016	B	Updates based on the kick-off meeting decisions	SC, JY, GC	DL
28 Oct 2016	C	H04 updated to countdown timer on and H05 for countdown timer off	SC	-
2nd Nov 2016	D	Updated cloud provision method via websocket	DL	DL

1 Overview


Murano Cloud
wss://
wss://
https://
Initial Provisioning
Device
Phone

1.1 Device Details

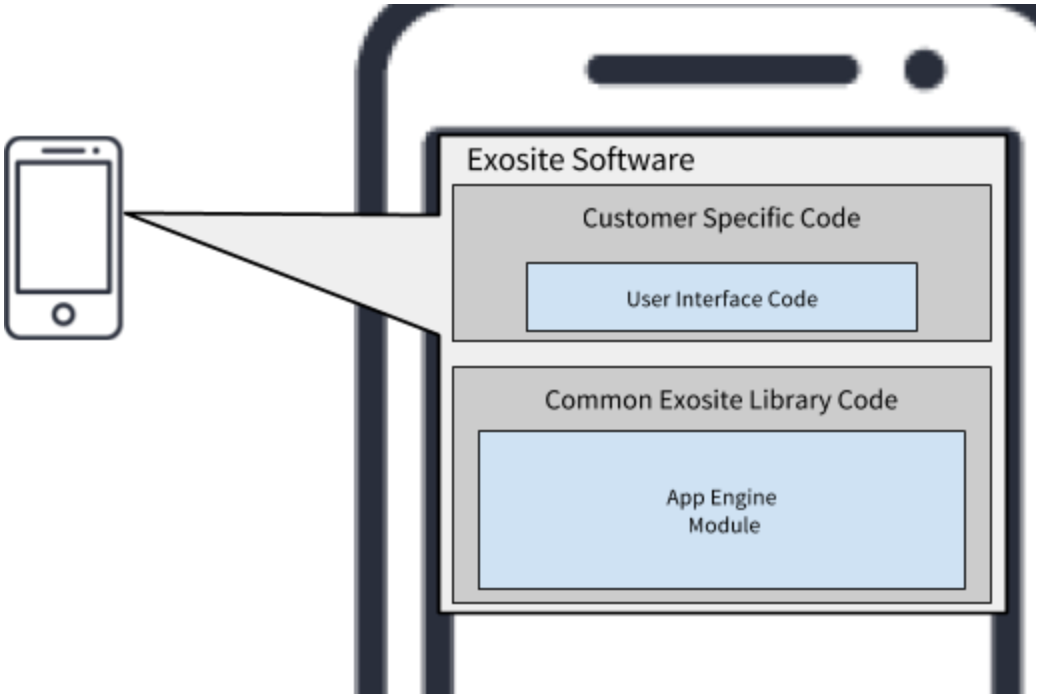


In the IoT Device itself the Main Controller Board is provided by the OEM. It is assumed that this board is already functioning and has a common HVAC feature set. The communication module on the other side will be stocked with Exosite firmware to cover the communication side.

The Wifi-Modules software is designed to be quickly adaptable by customers for different hardware environments. As a result the code is split into two major module categories:

1) Common Code
   These modules are expected to stay the same across many applications or just to require very minimal changes. It is assumed that 80% of the total code is in this common modules section.
2) Vertical Specific Code
   These modules are known to need changes per device hardware or device model configuration depending on whether the internal communication protocol is different (UART translation module) or whether the feature set that is supported varies (Smart Home Event Module)

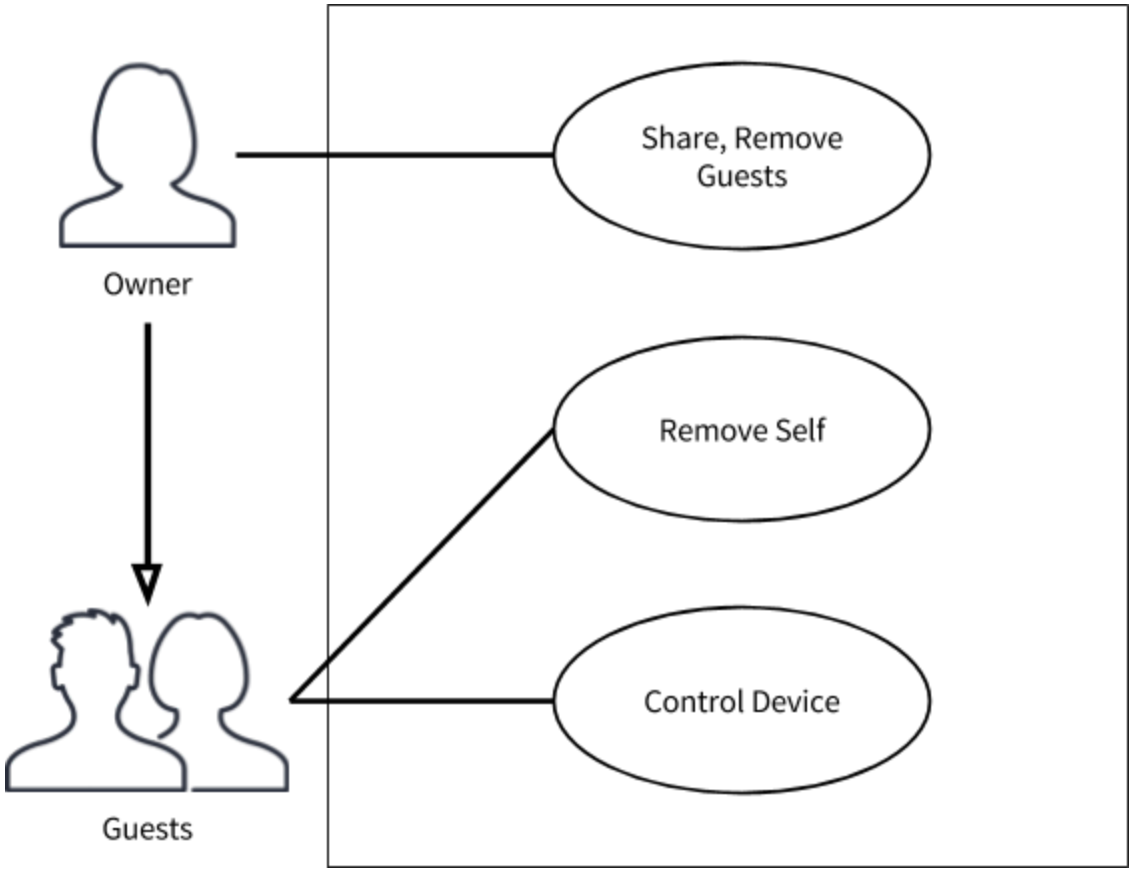
1.2 Phone Details



The complete phone software implementation is expected to come from Exosite and provided to customers for customization - with focus on the UI. For this reason the split between Vertical Specific Code and Common Code as in the device side is here even stronger. All UI code should be isolated in it's own physical files so it can be changed e.g. for branding without affecting any of the application logic.

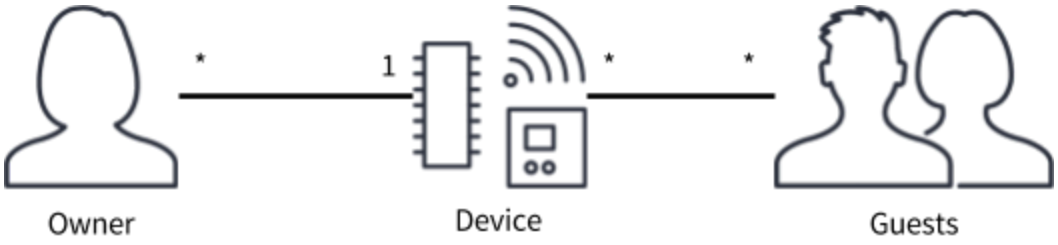
The business logic on the other side should stay the same 80%-90% of the time and designed to not care about UI changes.

1.3 User Permission Topology



Each device **can have one and only one "Owner"**. The owner can manage many guests for the device. The owner and guests can all control the device. Guests and Owner can also remove their own control.

Note: If the owner is removing his own control also all guests will loose control over the device. The device will become uncontrollable until it is physically reset and assigned to a new owner.



2 Terminology

2.1 Message Types

Throughout this specification there are three different types of messages:

2.1.1 Requests

Every request expects to receive a response. HTTP is by default always request response. For WebSockets requests will be encoded as:

```
{"id": <id>, "request": <request_name>, "data": <request_data>}
```

Specifically the <id> field is used to map requests to their responses as the WebSockets protocol is asynchronous by nature.

2.1.2 Responses

Responses when using WebSockets are encoded as:

```
{"id": <id>, "response": <response_name>, "status": <status>, "data":
<response_data>}
```

2.1.3 Events

Events do not require responses. This specification uses Events only from Cloud => Phone communication to inform of data changes. Events do not need responses. For WebSockets events are encoded as:

```
{"event": <event_name>, "data": <event_data>}
```

2.1.4 Example of color coding

	Request/Response (Command Line Bash)
Request	curl https://192.168.1.1:32051/info
Response	HTTP/1.1 200 OK Date: Mon, 30 Sep 2016 16:18:20 GMT Content-Length: 52 Connection: close Content-Type: text/plain; charset=utf-8 serial=<sn>&last_error=<last_error>
Error	HTTP/1.1 400 Bad Request

Response	Date: Mon, 30 Sep 2016 16:18:20 GMT Content-Length: 6 Connection: close Content-Type: text/plain; charset=utf-8 error!
Event	

2.2 Protocols

Name	Description	RFC
HTTP	HTTP(s) protocol is the simplest protocol for requests/response interaction. This protocol is used mostly during the provisioning phase. **Protocols:** https:// - TCP + TLS + HTTP	https://tools.ietf.org/html/rfc2616
TLS	Security and encrytpion layer used to protect HTTPS, WebSockets and CBOR streaming. **Protocols:** tls:// - TCP + TLS	https://tools.ietf.org/html/rfc5246
WebSocket	WebSockets are based on top HTTP and used between the Cloud and the Phone for asynchronous communication **Protocols:** wss:// - TCP + TLS + WEBSOCKET	https://tools.ietf.org/html/rfc6455
CBOR	CBOR is a JSON compatible efficient encoding. The main purpose for Exosite to use it is to safe bandwidth and memory on embedded devices. **Protocols:** cbors:// - TCP + TLS + CBOR	https://tools.ietf.org/html/rfc7049
JSON	JSON is the defacto standard encoding scheme in the internet. It is used broadly within this specification to define data semantic.	https://tools.ietf.org/html/rfc7159

The communication protocols are used in following phases are:

Phase 1: Provisioning	Mobile Phone => Device (initial provisioning)	https:// (Device is Server)
	Device => Cloud (initial provisioning)	wss:// (Murano is Server)

Phase 2: Data Exchange	Mobile Phone => Cloud	wss:// (Murano is Server)
	Device => Cloud (data flow)	wss:// (Murano is Server)

2.3 Simple Data Types

Name	Type	Example	Description
<event_name>, <request_name>	String - 20 bytes [a-z_]	"provision"	The event or request name according to the command reference.
<id>	Number or Alnum String	123 or "123ab"	A id number that is used to match request and response in WebSocket and CBOR communication.
<key>	Device Secret		
<product_id>, <pid>	String - 20 alnum bytes		Exosite identifier for the product
<pwd>	String	"s3cr3t"	Password of the Wifi-Network, different limits depending on wifi security.
<request_data>, <response_data>, <event_data>	String - Json	"data"	Data payload encoded in json for the data protocol
<role>	String Enum - ["guest", "owner"]	"guest"	Identifies a user level as guest user or owner of the device
<sec>	String - 4 alnum bytes	"open"	Can be one of ‘wep’, ‘wpa’, ‘wpa2’ or ‘open’ to choose the authentication mode of the target wifi
<secret>	String - 20 bytes hex	"ABCDEF1234ABCDEF1234"	Secret used for handshake
<sn>	String - 17 bytes hex	"AB:CD:EF:12:34:54"	MAC of the Wifi Chip
<ssid>	String - 32 alnum bytes	"ACCompany-AC001"	Service Set IDentifier

2.4 Complex Data Types (JSON Notation)

All complex data types are documented using JSON notation even though they will be transported both cbor and json respectively.

2.4.1 Calendar Type

The Calendar is used both on the phone application and in the device to define scheduling. It is represented as an array of individual calendar entries. There can be a **maximum of 10** calendar entries.

Calendar Example	Description
[	

<pre>{
"start": "12:24",
"end": "13:24",
"days": [1,2,3,4,5,6,7],
"active": 1,
"active_until":
1477377969,
"esh": {
"H00":1
}
}
]</pre>
	"start": Start time of the event in UTC
	"end": End time of the event in UTC
	"days": Mon=1,...,Sun=7 days the event should trigger
	"active": Allows enabling/disabling without deleting
	"active_until": UTC timestamp after which the event is invalid
	"esh": Object of all values to be set during the timeframe.
	Afer the event values are reset to their previous value.

3 Exosite ESH Protocol Definitions

The exosite smart home (esh) protocol definitions define different device types and semantic for values types.

Name	Example	Description
class	0	0: HouseHold Devices
device_id	'F'	Describes the type of device. This will also enable/disable different command sets per device. 0: General (通用) F: Fan (電扇)
fields	["H00"]	Fields are defined per device family. Each device_id family has a different set of definitions.
esh_version	"4.00"	The current version of the ESH protocol, the current version is "4.00"
brand	"Company"	The name of the device offering company.
model	"AX4200"	The name of this product model.

3.1 Fields for device_id = F (Fan)

Field Name	Value Range	R/W	Description
H00	0: Off 1: On	R+W	On/Off
H01	0: Mode 0 1: Mode 1 Natural wind 2: Mode 2 3: Mode 3	R+W	Fan Mode
H02	0: Automatic Speed 1-32: Speed level	R+W	Fan Speed
H03	Min: -128 Max: 127	R	Indoor Temperature Celsius
H04	Min: 0 Max: 2359	R+W	Timer on This specifies the time of the day on which the fan should **enable** itself. The time encoded as decimal. E.g. 0100 means 1am 1330 means 1:30pm and so on
H05	Min: 0 Max: 2359	R+W	Timer off This specifies the time of the day on which the fan should **disable** itself. The time encoded as decimal.
H06	0: Forward 1: Reverse	R+W	Fan Rotation
H07	Min: 0 Max: 1440	R	Current in 0.1 Amp units
H08	Min: 0 Max: 1440	R	Voltage in V
H09	Min: 0 Max: 1440	R	Power factor
H10	Min: 0 Max: 1440	R	Power in W
H0a	Min: 0 Max: 1440	R	kwh in 0.1 kWh units
H0b	0: Off 1: On	R+W	Light
H0c	Min: 0 Max: 100	R+W	Light dimming % 100: Means full brightness 0: Mean light is off

4 Provisioning Overview

Provisioning will be done local in the current wifi the flow is like this:

1) Device is new / or reset
2) Device goes into AP mode
3) Phone app is opened for the first time
4) Phone app asks for user to login / registration
5) Phone-User can choose to sign up via Email or Facebook
6) Phone Signup and login finish
7) Phone looks and find Device in AP mode
   SSID is hardcoded to "<Vendor>-<Model>"
8) Phone send provisioning details (wifi-ssid and wifi-password) via https
   GET 192.168.1.1/provision?ssid=<ssid>&pwd=<pwd>&sec=wep/wpa/wpa2/open
9) Device responds to info request
10) Device answers with secret and serial number in the body
    serial=<sn>& secret=<secret>
11) Device tries wifi credentials
12) Phone connects to cloud and sends custom provision request:
    wss://<smarthome>.exosite.com/ws => "PROVISION:<secret>"
13) Device sends device secret to Exosite Murano
14) Device starts broadcasting its serial number via mdns
15) Cloud receives device write event
    If no phone is found cloud stores data into key-value store list for later retrieval
16) Cloud assigns user owner-level-rights to this device for the waiting phone user.
17) Phone receives message about connected device.
18) Phone can now control the device until it is reset.

De-Provisioning (when the device was owned by somebody else):

1) The device is reset
2) Device goes into AP mode (wifi-direction optional)
3) … provisioning by the new user ...
4) The cloud receives "secret"
5) Cloud finds existing user with user owner-level-rights who used a different secret
6) Cloud deletes rights for existing users and assigns new user owner-level-rights

Locality Detection:

1) The device is broadcasting its serial number via mdns