# A Tale of Overengineering

Implementing the SpaceAPI in Rust

Danilo Bargen (@dbrgn), Raphael Nestler (@rnestler)

June 17, 2017

Coredump Rapperswil

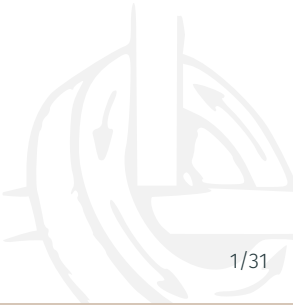
# Outline

1. Quick reminder - What is the SpaceAPI?
2. The Beginning
3. Rewrite it in Rust!
4. Future

# Quick reminder - What is the SpaceAPI?

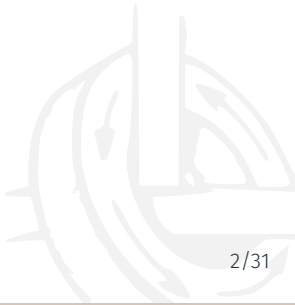
# What is the SpaceAPI

- A JSON schema
- Describes your hackerspace
- Mostly static but may contain dynamic data



# SpaceAPI Excerpt

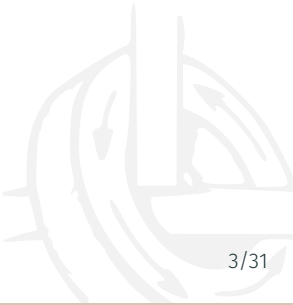

```
{
    "api": "0.13",
    "contact": {
        "email": "vorstand@lists.coredump.ch",
        "twitter": "@coredump_ch"
    },
    "sensors": {
        "people_now_present": [
            {
                "location": "Hackerspace",
                "value": 0
            }
        ]
    },
    "state": {
        "message": "Open Mondays from 20:00",
        "open": false
    }
}
```

# The Beginning

# Requirement

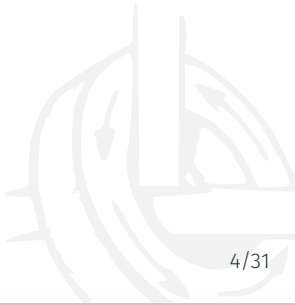
- Serve SpaceAPI JSON data
- Mostly static except state, sensors and events
- Endpoint to update sensor data



# A 100 Line Python Script

- Our first implementation was ~100 lines of Python[1]
- Hacky but worked

[1] https://github.com/coredump-ch/status/tree/1ae92a58c13b5ee106671092717f1bf4b4b9ee97

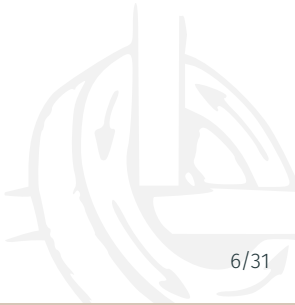


# A 100 Line Python Script

```
# -*- coding: utf-8 -*-
import json

from bottle import default_app, route, response, run, request


app = default_app()


def get_number_of_people():
+-- 10 lines: """----------------------------------------------------------------


@route('/')
def json_out():
+-- 59 lines: response.set_header('Access-Control-Allow-Origin', '*')----------


@route('/update', method='POST')
def update():
+--  8 lines: """----------------------------------------------------------------


@route('/sensors/people_now_present/html')
def sensor_people_html():
+-- 12 lines: """----------------------------------------------------------------


if __name__ == '__main__':
    run(host='localhost', port=8080)
```

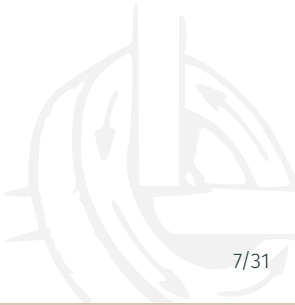
# Reading the sensor...

```
def get_number_of_people():
    """
    Return an integer or None.
    """
    people = None
    try:
        with open('people.txt', 'r') as f:
            people = int(f.read().strip())
    except:
        pass
    return people
```



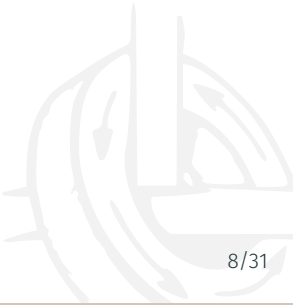
# Updating the sensor...

```
@route('/update', method='POST')
def update():
    """
    Update the data in a text file.
    TODO: Public / Private key crypto.
    """
    people = int(request.POST.get('people'))
    with open('people.txt', 'w') as f:
        f.write(str(people))
    return 'OK'
```



# Hacky, but actually worked!

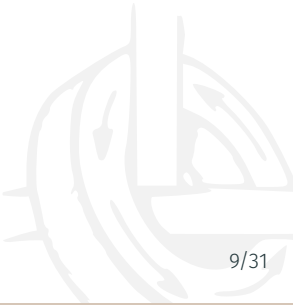
- Hacky but worked
- Probably a race condition with the file access?
- Ugly to extend with further sensors



# Rewrite it in Rust!

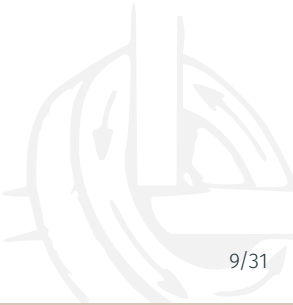
# Why?

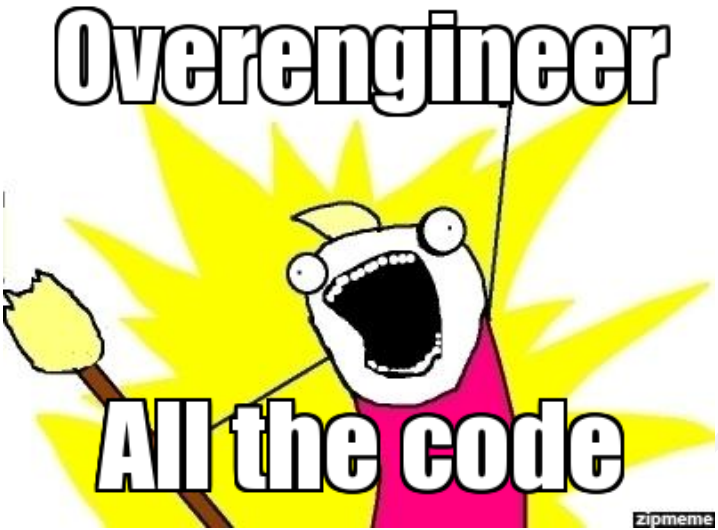
- If it worked why rewrite it?



# Why?

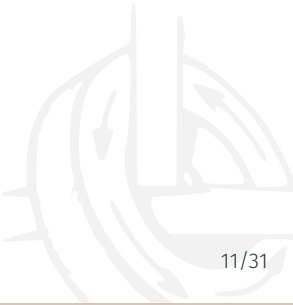
- If it worked why rewrite it?
- Learn Rust
- Have a reasonable sized project




Overengineer
All the code
zipmeme

# Goals

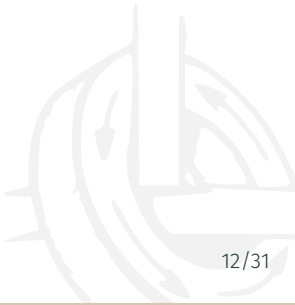
- Encode the SpaceAPI rules in the type system
  $\rightarrow$ Impossible to generate non conforming JSON
- Use a better backend for data storage
- Make it reusable for other hackerspaces



# Current Status

- spaceapi-rs[2]
  - SpaceAPI schema encoded in the type system
  - Serialization / Deserialization
- spaceapi-server-rs[3]
  - Server implemented with Iron
  - Reads sensor values from Redis DB
  - Allows updating sensor values
- status[4]
  - Implementation for coredump

[2] `https://github.com/coredump-ch/spaceapi-rs`
[3] `https://github.com/coredump-ch/spaceapi-server-rs`
[4] `https://github.com/coredump-ch/status`



## Rewrite it in Rust!

spaceapi-rs

# spaceapi-rs

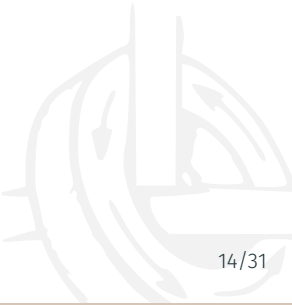
- spaceapi-rs[5]
  - SpaceAPI schema encoded in the type system
  - Serialization / Deserialization

- spaceapi-server-rs[6]
  - Server implemented with Iron
  - Reads sensor values from Redis DB
  - Allows updating sensor values
- status[7]
  - Implementation for coredump

[5] https://github.com/coredump-ch/spaceapi-rs
[6] https://github.com/coredump-ch/spaceapi-server-rs
[7] https://github.com/coredump-ch/status

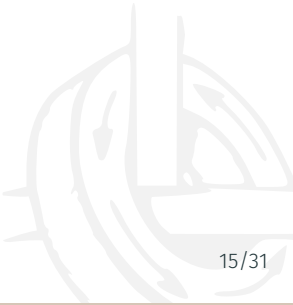
# Data Types

```
pub struct Status {
    pub api: String,
    pub space: String,
    pub logo: String,
    pub url: String,
    pub location: Location,
    pub contact: Contact,
    ...
}
```



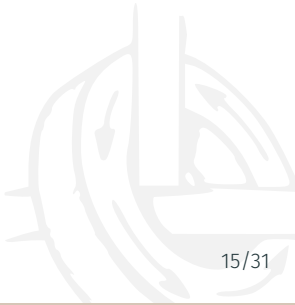
# Difficulties

- SpaceAPI specs don't allow null for optional entries
- `rustc_serialize` serialized 'Option<T>' to null



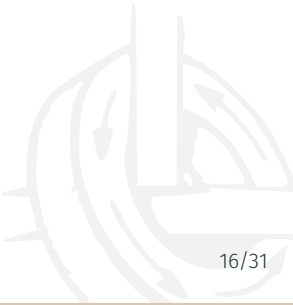
# Difficulties

- SpaceAPI specs don't allow null for optional entries
- `rustc_serialize` serialized 'Option<T>' to null
  $\rightarrow$ We rolled our own serialization with an 'Optional<T>' type
- Serde solved this quite elegant



# Serde

```rust
#[derive(Serialize, Deserialize, ...)]
pub struct Location {
    #[serde(skip_serializing_if = "Option::is_none")]
    pub address: Option<String>,
    pub lat: f64,
    pub lon: f64,
}
```



```rust
#[derive(Serialize, Deserialize, ...)]
pub struct Location {
    #[serde(skip_serializing_if = "Option::is_none")]
    pub address: Option<String>,
    pub lat: f64,
    pub lon: f64,
}

#[derive(Serialize, Deserialize, ...)]
pub struct Sensors {
    #[serde(default, skip_serializing_if = "Vec::is_empty")]
    pub people_now_present: Vec<PeopleNowPresentSensor>,
    #[serde(default, skip_serializing_if = "Vec::is_empty")]
    pub temperature: Vec<TemperatureSensor>,
}
```

# Find the problem

## events (array of object)

▼ Details

Events which happened recently in your space and which could be interesting to the public, like 'User X has entered/triggered/did something at timestamp Z'

**Nested array items**

### name (string) required

▶ Details

### type (string) required

▶ Details

### timestamp (number) required

▶ Details

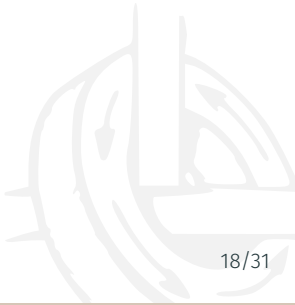
### extra (string)

▶ Details

# Keywords...

Rust doesn't like keywords as identifier...

```
error: expected identifier, found keyword `type`
  --> src/status.rs:52:9
   |
52 |     pub type: String,
   |         ^^^^
```



# Serde to the rescue!

```rust
#[derive(Serialize, Deserialize, ...)]
pub struct Event {
    pub name: String,
    #[serde(rename = "type")]
    pub type_: String,
    pub timestamp: u64,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub extra: Option<String>,
}
```

## Rewrite it in Rust!

spaceapi-server-rs

# Server

- spaceapi-rs[8]
  - SpaceAPI schema encoded in the type system
  - Serialization / Deserialization

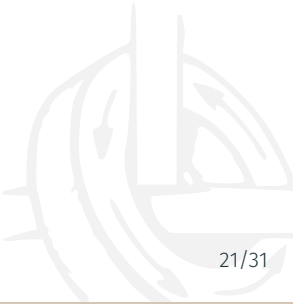
- spaceapi-server-rs[9]
  - Server implemented with Iron
  - Reads sensor values from Redis DB
  - Allows updating sensor values

- status[10]
  - Implementation for coredump

[8] https://github.com/coredump-ch/spaceapi-rs
[9] https://github.com/coredump-ch/spaceapi-server-rs
[10] https://github.com/coredump-ch/status

# Server

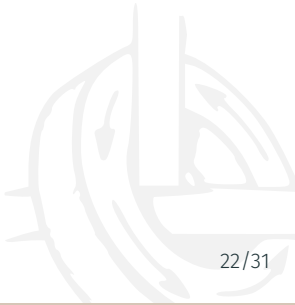
- Webframework: iron[11]; Maps HTTP nicely to the type system
- DB: r2d2-redis[12]
- Should be usable by every hackerspace



[11] http://ironframework.io/
[12] https://github.com/sorccu/r2d2-redis

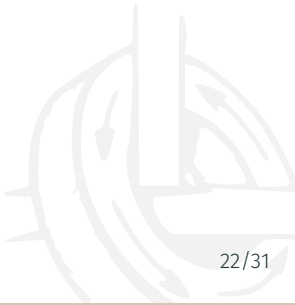
# Dynamic Values

- Our SpaceAPI types do not allow missing data. How do we handle dynamic data from the DB?



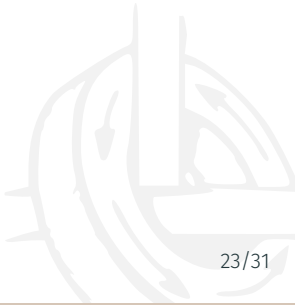
# Dynamic Values

- Our SpaceAPI types do not allow missing data. How do we handle dynamic data from the DB?
  → `SensorTemplate`



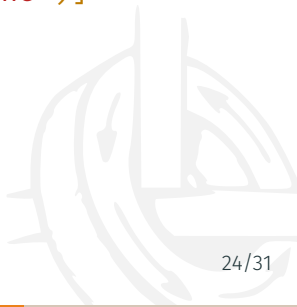
# SensorTemplate

```rust
/// A trait for all possible sensor templates.
///
/// A sensor template is like a sensor struct, but without the actual data in i
/// A `SensorTemplate` is capable of registering itself in a `Sensors` struct.
pub trait SensorTemplate : Send+Sync {
    fn to_sensor(&self, value_str: &str, sensors: &mut Sensors);
}
```



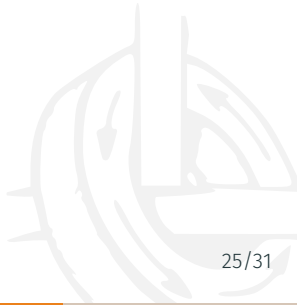
# An Example

```rust
pub struct PeopleNowPresentSensor {
    #[serde(skip_serializing_if = "Option::is_none")]
    pub location: Option<String>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub name: Option<String>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub names: Option<Vec<String>>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub description: Option<String>,
    pub value: u64,
}
```



# An Example

```
pub struct PeopleNowPresentSensorTemplate {
    pub location: Option<String>,
    pub name: Option<String>,
    pub names: Option<Vec<String>>,
    pub description: Option<String>,
}
```



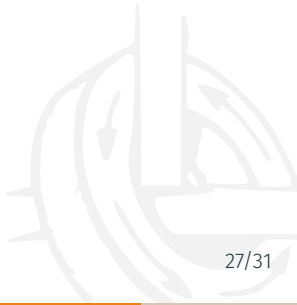
# Implementing SensorTemplate

```
impl SensorTemplate for PeopleNowPresentSensorTemplate {
    fn to_sensor(&self, value_str: &str, sensors: &mut Sensors) {
        if value_str.parse::<u64>().map(|value|{
            let sensor = PeopleNowPresentSensor {
                location: self.location.clone(),
                name: self.name.clone(),
                names: self.names.clone(),
                description: self.description.clone(),
                value: value,
            };

            sensors.people_now_present.push(sensor);
        }).is_err() {
            warn!("Could not parse value '{}', omiting PeopleNowPresentSensor",
        }
    }
}
```

# StatusModifier

To further adapt the server to your needs

```rust
/// `StatusModifier`s are used to modify the status
pub trait StatusModifier: Send + Sync {
    /// Called after all registered sensors are read
    fn modify(&self, status: &mut api::Status);
}
```



## Rewrite it in Rust!

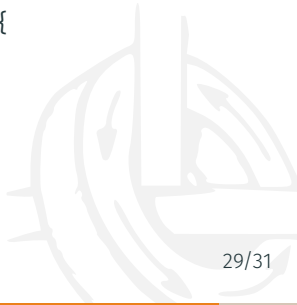
Implement your own!

# Goals

- Both `spaceapi-rs` and `spaceapi-server-rs` are fully reusable libraries
- Customizing happens at compile time

# Implement your own!

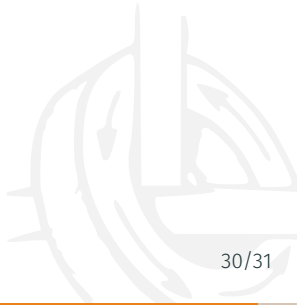
```
let status = StatusBuilder::new("coredump")
    .logo("https://www.coredump.ch/logo.png")
    .url("https://www.coredump.ch/")
        ...
    .build().expect("Creating status failed!");

let mut server = SpaceapiServer::new("128.0.0.1:8000", status, "redis://127.0.0
                                     vec![Box::new(StateFromPeopleNowPresent)])
                                  .expect("Could not initialize server");

server.register_sensor(Box::new(TemperatureSensorTemplate {
    unit: "°C".into(),
    location: "Hackerspace".into(),
    name: Value("Raspberry CPU".into()),
    description: Absent,
}), "temperature_raspi".into());
```



# Future

# What are our plans?

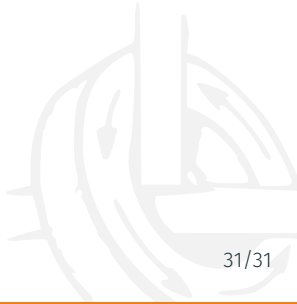
- Complete spaceapi-rs (Lots of missing API keys, contributors welcome!)
- Create reusable server binary (loads static info from file at runtime) for other hackerspaces to use
- ...



# Wanna help?

We collect easy issues on both spaceapi-rs and spaceapi-server-rs

- `https://github.com/coredump-ch/spaceapi-rs/issues?q=is%3Aissue+is%3Aopen+label%3Aeasy`
- `https://github.com/coredump-ch/spaceapi-server-rs/issues?q=is%3Aissue+is%3Aopen+label%3Aeasy`



# Thank you!

www.coredump.ch

Slides: https://github.com/coredump-ch/cosin-2017-talks