Rust in a UniFFIed World

About me

- Firefox Telemetry engineer at Mozilla
- Rust Community & Core Team member
- Scuba diver

Twitter: @badboy_
Blog: fnordig.de

Slides:

fnordig.de/talks/2022/rustnigeria/slides.pdf

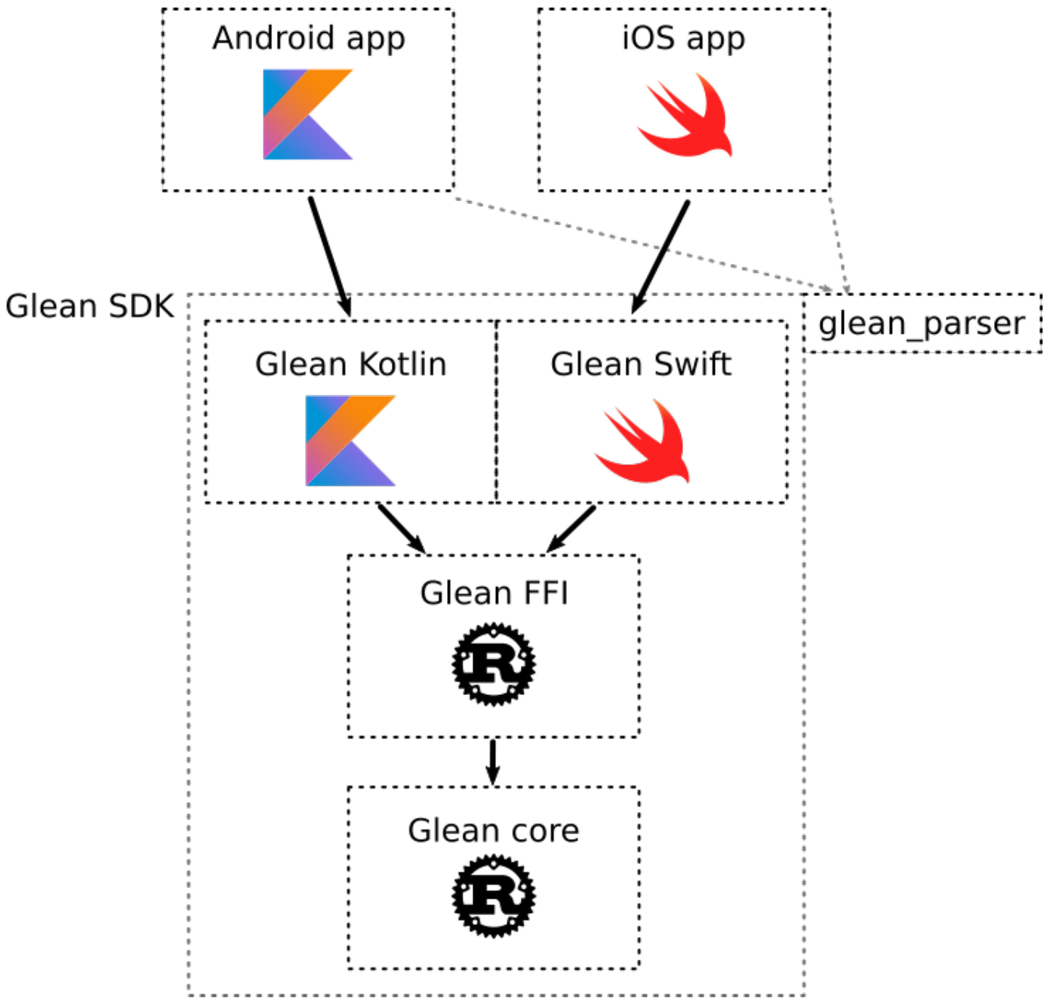
https://github.com/mozilla/glean

Collecting Data Responsibly and at Scale [2]

[2] StarCon 2019 Talk by chutten.

Glean SDK stack

- low-level Rust library for the logic
- Swift for iOS targets
- Kotlin for Android targets
- Python for server and tooling
- C++ and JavaScript for Firefox Desktop
- Rust for others



What if we don't want to write code for every platform?

UniFFI[3]

a multi-language bindings generator

[3] github.com/mozilla/uniffi-rs

Define your interface

```
// WebIDL-like interface definition language
interface Rocket {
    constructor(string name);

    void lock_steering(string direction);

    [Throws=LaunchError]
    boolean launch();
};
```

Implement it in Rust[4]

```
struct Rocket {
  name: String,
  direction: String,
}

impl Rocket {
  fn new(name: String) -> Rocket {
    Rocket {
      name: name, steering: "".into()
    }
  }
```

```
fn lock_steering(&mut self, dir: String) {
  self.direction = dir;
}

fn launch(&self) -> Result<bool> {
  if self.direction != "up" {
    return Err(LaunchError::RocketLaunch);
  }
  Ok(true)
}
```

[4] Full example: github.com/badboy/rocketscience

✨ 🪄 MAGIC! 🪄 ✨

Use it in Swift

```
import rocketscience

let rocket = Rocket(name: "Orbiter")
rocket.lockSteering("up")
try! rocket.launch()
```

Use it in Kotlin

```
import rediger.uniffi.rocketscience.*

val rocket = Rocket(name = "Orbiter")
rocket.lockSteering("up")
rocket.launch()
```

Use it in Python

```
from rocketscience import *

rocket = Rocket(name = "Orbiter")
rocket.lock_steering("up")
rocket.launch()
```

UniFFI ✨ 🪄 magic! 🪄 ✨

- Generates the FFI layer for you
- Takes care of type conversions on both sides
- Convenient APIs for the target language

UniFFI under the hood (Rust)[5]

```rust
#[no_mangle]
pub extern "C" fn ffi_rocketscience_b310_Rocket_object_free(ptr: *const c_void) {
    assert!(!ptr.is_null());
    drop(unsafe { std::sync::Arc::from_raw(ptr as *const Rocket) })
}

#[no_mangle]
pub extern "C" fn rocketscience_b310_Rocket_new(name: uniffi::RustBuffer) -> *const c_void {
    let _new = Rocket::new(<String as uniffi::FfiConverter>::try_lift(name));
    let _arc = std::sync::Arc::new(_new);
    <std::sync::Arc<Rocket> as uniffi::FfiConverter>::lower(_arc)
}
```

[5] rocketscience.uniffi.rs

UniFFI under the hood (Swift)[6]

```
public class Rocket: RocketProtocol {
    public convenience init(name: String) {
        self.pointer = rocketscience_b310_Rocket_new(name.lower())
    }

    deinit { ffi_rocketscience_b310_Rocket_object_free(pointer) }

    public func lockSteering(direction: Direction) {
        rocketscience_b310_Rocket_lock_steering(self.pointer, direction.lower())
    }
}
```

[6] rocketscience.swift

We wrote code like that before!

We wrote code like that before!

 Computers are better at it.

UniFFI Design Principle 1:

Safety first

UniFFI Design Principle 2:

Performance is a feature, but not a deal-breaker

UniFFI Design Principle 3:

Produce bindings that feel idiomatic for the target language

Links

- Slides: fnordig.de/talks/2022/rustnigeria/slides.pdf
- Rocket Science example: github.com/badboy/rocketscience
- UniFFI docs: mozilla.github.io/uniffi-rs
- Glean SDK repository: github.com/mozilla/glean
- Glean SDK docs: mozilla.github.io/glean
- Mozilla Data blog: blog.mozilla.org/data
- me on Twitter: @badboy_

Questions?

Questions?

Twitter: @badboy_

Mail: janerik@fnordig.de