Leveraging Rust to build cross-platform mobile libraries

About me

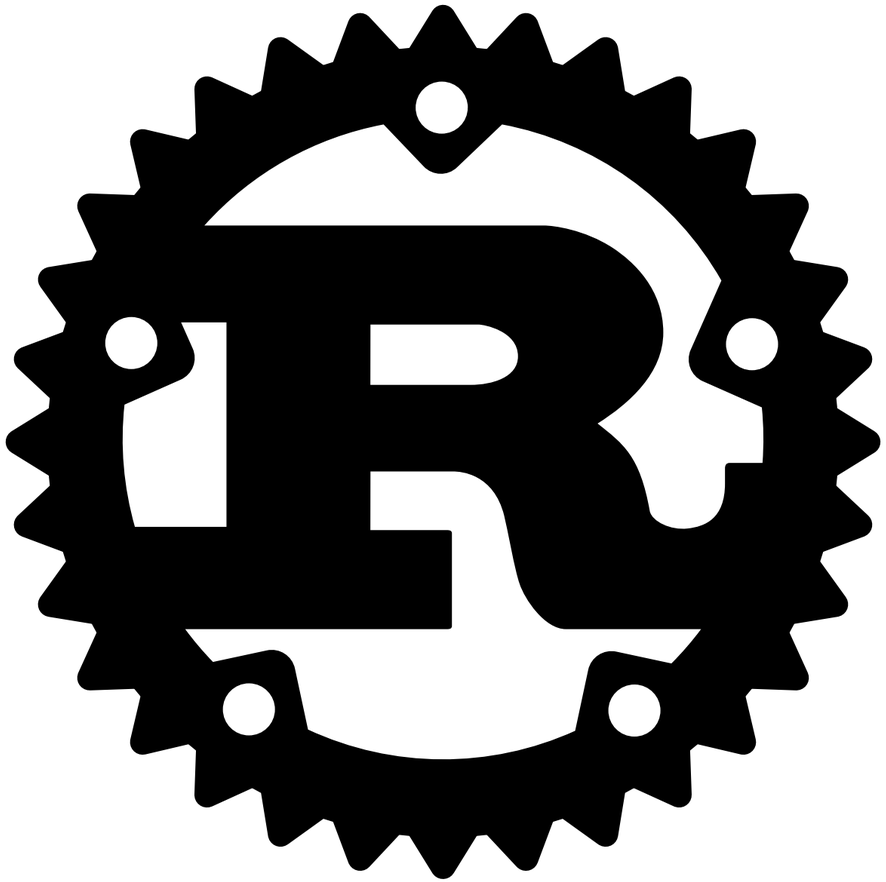
- Firefox Telemetry engineer at Mozilla
- Rust Community Team member
- Scuba diver

Twitter: @badboy_
Blog: fnordig.de

Slides:

fnordig.de/talks/2020/rustydays/slides.pdf


RUSTFEST
BARCELONA 2019



GLEAN

telemetry for humans

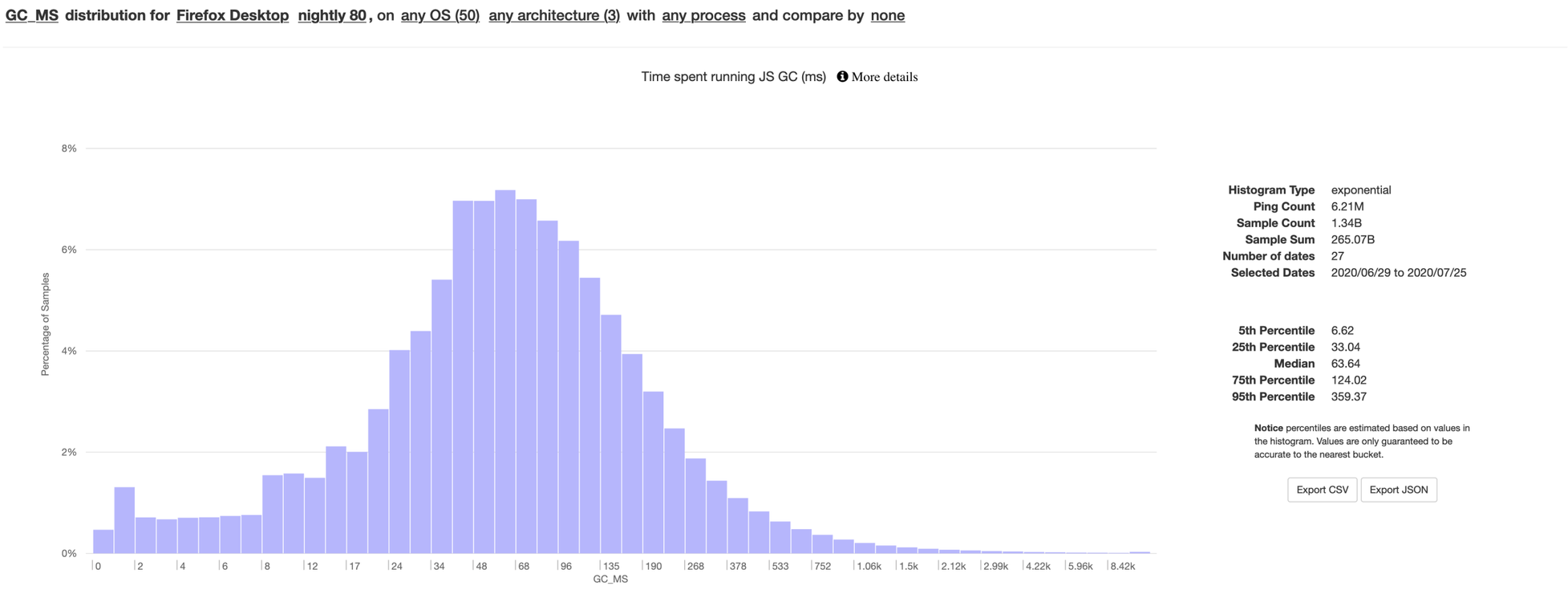
Firefox Telemetry

Firefox Telemetry

A quick overview

1. Performance **metrics** for our products
2. Packaged in **pings** sent at controlled schedules
3. Following our **Lean Data Practices**

Metric: Time spent running the JS GC[1]



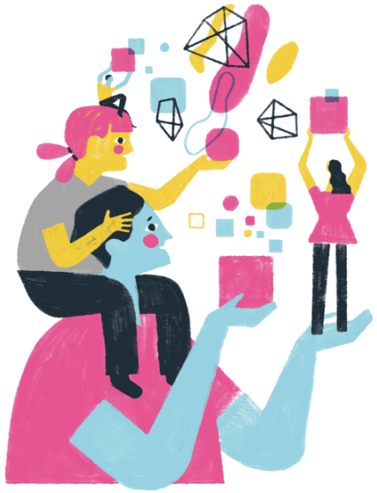
[1] Measurement Dashboard at https://telemetry.mozilla.org/new-pipeline/dist.html

Lean Data Practices

The Three Principles

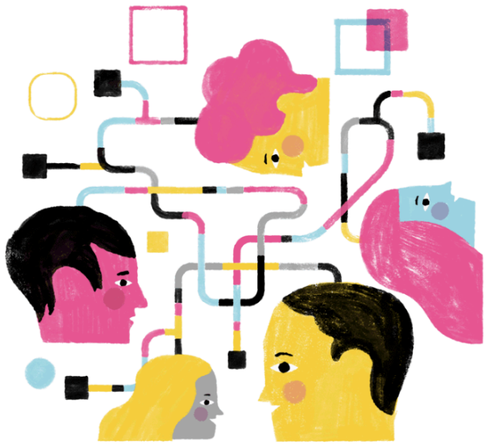
Stay Lean

Decide if all your data collection delivers value.



Build Security

Learn how to protect customer data.



Engage Your Users

Keep customers informed and empowered.

Collecting Data Responsibly and at Scale [2]

[2] StarCon 2019 Talk by chutten.

Geoffroy Couprie
@gcouprie

Describe your software in the most boring way possible
I'll start: memcpy over the network

12:18 PM · Jul 7, 2020

524 343 people are Tweeting about this

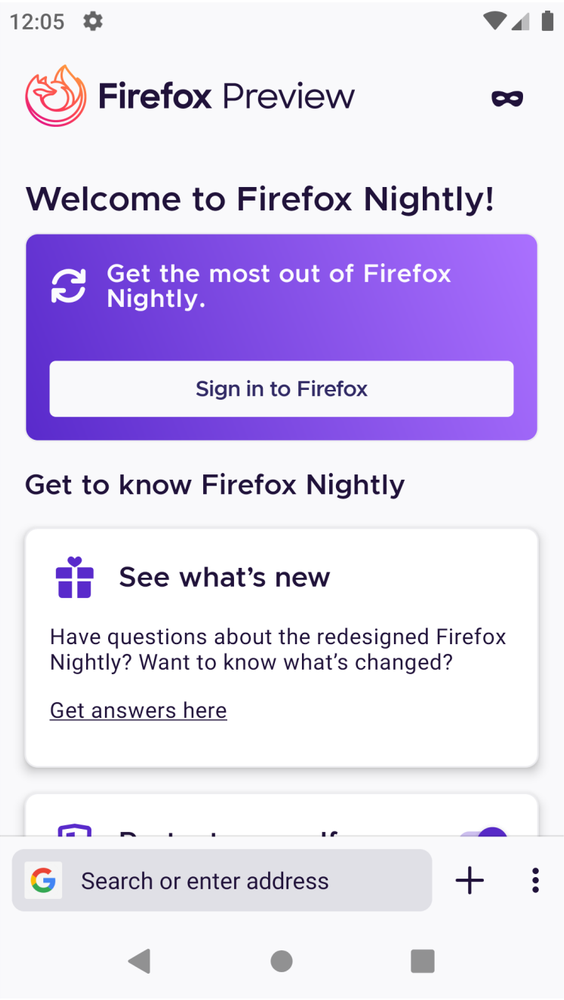
jan-erik
@badboy_

storing some integers and sending them JSON-encoded to a server.

11:10 AM · Jul 21, 2020

1 See jan-erik's other Tweets

Firefox for Android



The Glean SDK is a modern approach for a Telemetry library and is part of the Glean project.

Introducing Glean — Telemetry for humans
by Georg Fritzsche.

Telemetry API in Firefox Desktop

```
Services.telemetry.scalarAdd(
        "browser_engagement.max_concurrent_tab_count",
        1
);
```

Scalars.yaml

```yaml
browser.engagement:
 max_concurrent_tab_count:
   bug_numbers:
     - 1271304
   description: >
     The count of maximum number of tabs open during a subsession,
     across all windows, including tabs in private windows and restored
     at startup.
   expires: "81"
   kind: uint
   notification_emails:
     - someone@mozilla.com
   release_channel_collection: opt-out
   products:
     - 'firefox'
   record_in_processes:
     - 'main'
```

Telemetry API in Firefox Desktop

```
Services.telemetry.scalarAdd(
        "browser_engagement.max_concurrent_tab_count",
        1
);
```

```
BrowserEngagement.max_concurrent_tab_count.add(1)
```

```
BrowserEngagement.max_concurrent_tab_count.add(1)

|---------------|                                |--|
 Category object                                 integer

                 |-------------------------|
                    Counter metric
                                            |---|
                                             increment function
```

New Telemetry requirements

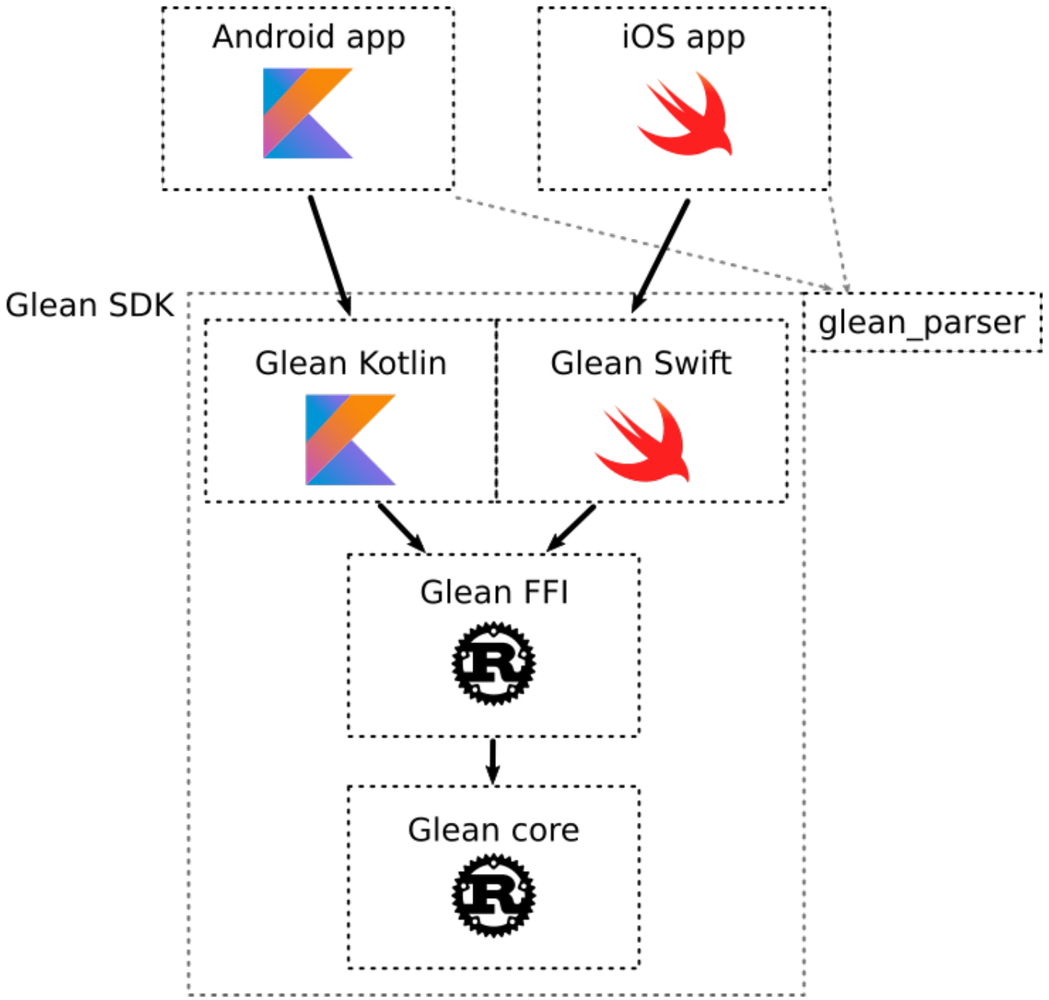
- Declarative definitions of metrics
- Share core implementation cross-platform
- Ergonomic API per target language

GLEAN

telemetry for humans

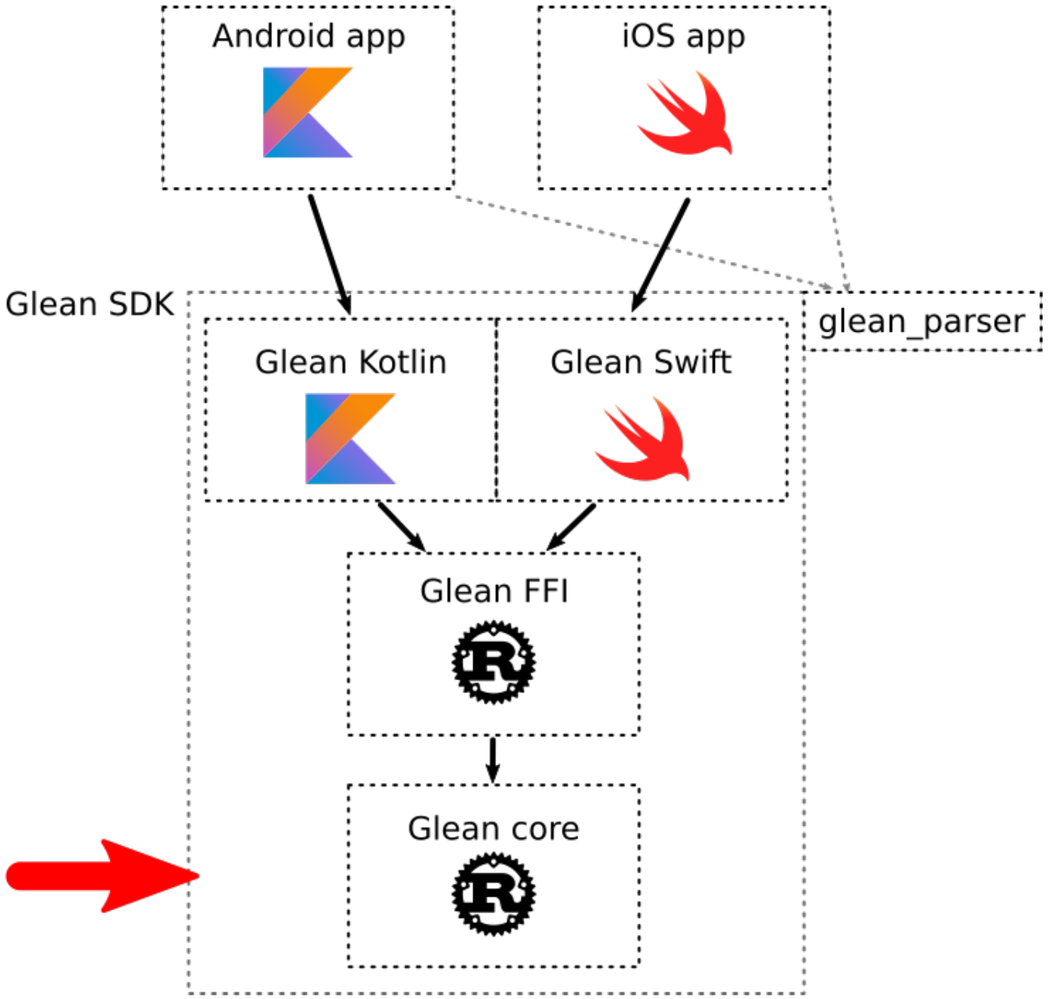
https://github.com/mozilla/glean

Glean SDK stack



Glean Core

A Rust crate



Glean Core - a Rust crate

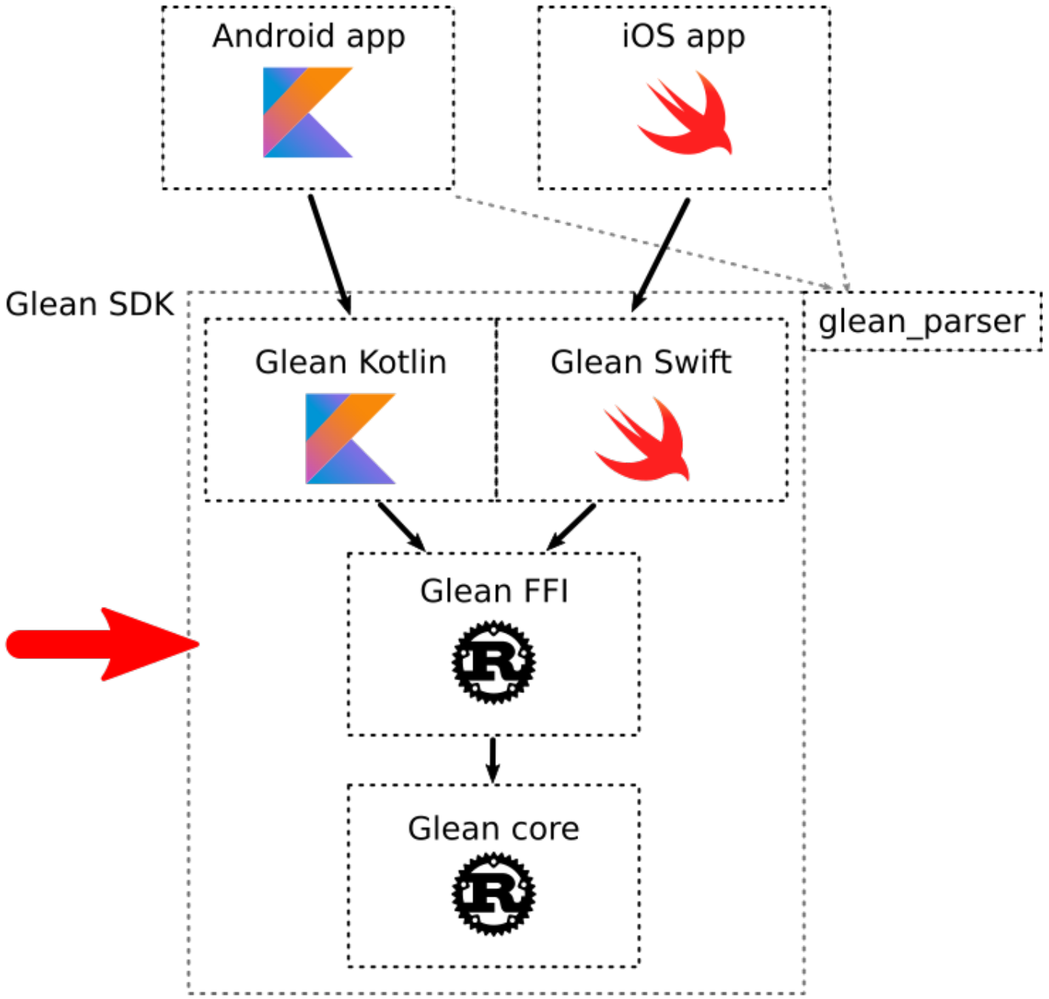
```
#[derive(Debug)]
struct Glean {
    data_path: PathBuf,
    upload_enabled: bool,
    data_store: Database,
    event_data_store: EventDatabase,
    core_metrics: CoreMetrics,
    // ...
}
```

Metrics implemented as Rust types

```
struct CounterMetric {
    meta: CommonMetricData,
}

impl CounterMetric {
    fn add(&self, glean: &Glean, amount: i32) {
        glean
            .storage()
            .record_with(&self.meta, |old_value| old_value.add(amount))
    }
}
```

Glean FFI

the connection



Foreign **F**unction **I**nterface

Calling C functions

```
extern {
    fn c_rusty_days(days: c_int);
}

// ..

unsafe {
    c_rusty_days(30);
}
```

Getting called from C

```
#[no_mangle]
pub extern "C" fn hello_rusty_days(data: c_int) -> *const u8 {
    "Rusty Days!\0".as_ptr()
}
```

"Rusty Days!\0"

CString to the rescue

```
let s = CString::new("Rusty days!").unwrap();
assert!(s.into_bytes_with_nul() == b"Rusty days!\0");
```

cbindgen

cbindgen creates C headers for Rust libraries which expose a public C API.

```
#[no_mangle]
pub extern "C" fn hello_rusty_days(data: c_int)
  -> *const u8 {
    "Rusty Days!\0".as_ptr()
}
```

```
#include <stdint.h>
#include <stdlib.h>

const uint8_t *hello_rusty_days(int data);
```

ffi-support

Support library to simplify implementing FFI libraries*.

** as done by application-services[3] & Glean*

[3] https://github.com/mozilla/application-services

ffi-support: `IntoFFI`

Convert Rust types into FFI-compatible types

```
unsafe trait IntoFfi: Sized {
    type Value;
    fn ffi_default() -> Self::Value;
    fn into_ffi_value(self) -> Self::Value;
}

unsafe impl IntoFfi for String {
    type Value = *mut c_char;

    // ...
}
```

ffi-support: `FfiStr`

A safe wrapper around a null-terminated string.

```
pub struct FfiStr<'a> { /* fields omitted */ }

#[no_mangle]
extern "C" fn hello_rusty_days(data: FfiStr) {
    // Use of `data` after this function returns is impossible
}
```

ffi-support: `ConcurrentHandleMap`

A locked map with handles to use across the FFI.

```
static COUNTER: Lazy<ConcurrentHandleMap<CounterMetric>>
    = Lazy::new(ConcurrentHandleMap::new);


extern "C" fn glean_new_counter_metric(name: ffi_support::FfiStr) -> u64 {
    COUNTER.insert_with_log(|| {
        Ok(glean_core::metrics::CounterMetric::new(name.as_str()))
    })
}
```

Compile Targets

```
$ rustup target list
aarch64-apple-ios (installed)
aarch64-fuchsia
aarch64-linux-android (installed)
aarch64-pc-windows-msvc
aarch64-unknown-linux-gnu
aarch64-unknown-linux-musl
aarch64-unknown-none
[...]
x86_64-apple-darwin (installed)
x86_64-apple-ios (installed)
[...]
x86_64-unknown-redox
```

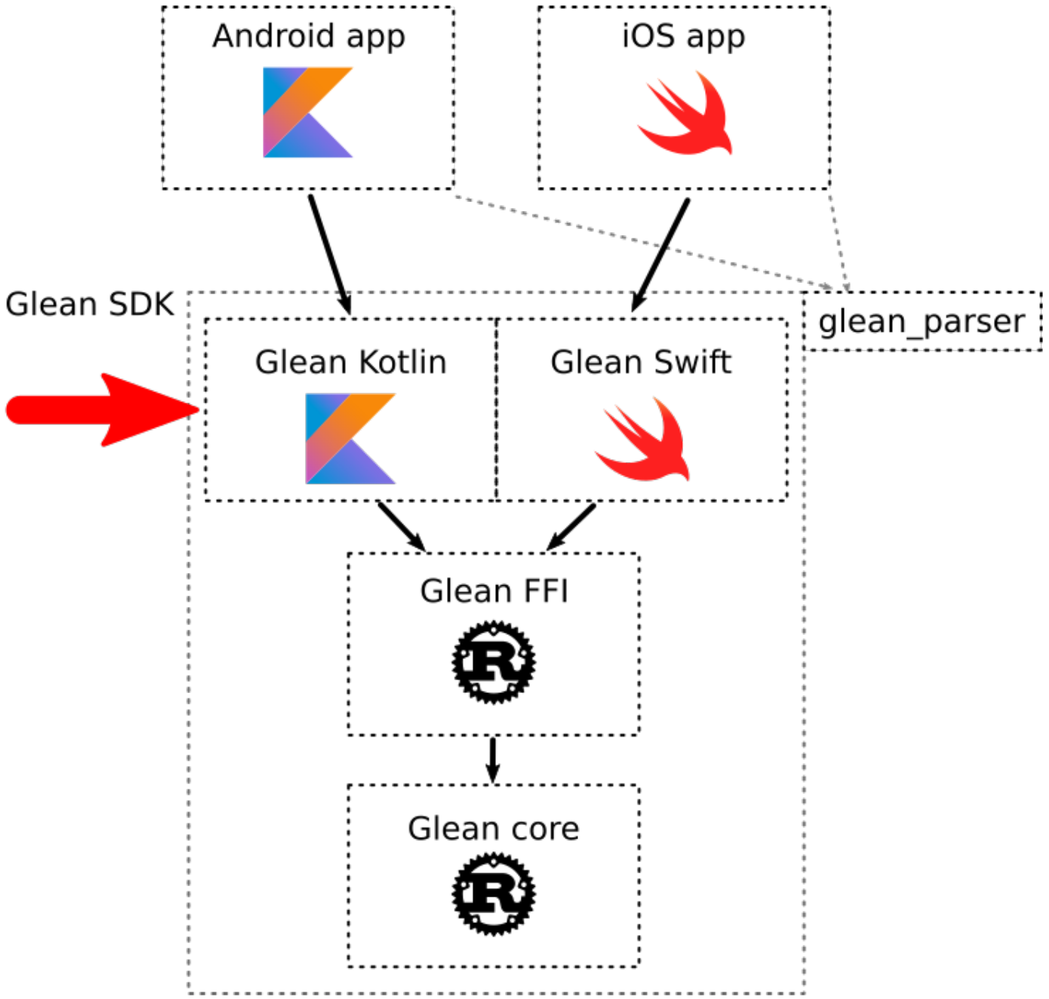
```
<arch><sub>-<vendor>-<sys>-<abi>
```

Glean targets

```
rustup target add armv7-linux-androideabi   # for arm
rustup target add i686-linux-android        # for x86
rustup target add aarch64-linux-android     # for arm64
rustup target add x86_64-linux-android      # for x86_64 (& simulator)
rustup target add x86_64-unknown-linux-gnu  # for linux-x86-64
rustup target add x86_64-apple-darwin       # for macOS
rustup target add x86_64-pc-windows-gnu     # for win32-x86-64-gnu
rustup target add x86_64-pc-windows-msvc    # for win32-x86-64-msvc
rustup target add aarch64-apple-ios         # iOS (actual devices)
rustup target add x86_64-apple-ios          # iOS simulator
```

Glean Kotlin

The Kotlin implementation



JNI is the Java Native Interface. It defines a way for the bytecode that Android compiles from managed code to interact with native code.

— from Android - JNI tips

Hello World with JNI

Rust

```
#[no_mangle]
extern "system" fn Java_HelloWorld_hello(
    env: JNIEnv,
    _class: JClass,
    input: JString,
) -> jstring {
  // ...
}
```

Java

```
class HelloWorld {
    static native String hello(String input);

    static {
        System.loadLibrary("mylib");
    }
}
```

Otavio Pace: Interop with Android, IOS and WASM in the same project

JNA provides Java programs easy access to native shared libraries without writing anything but Java code - no JNI or native code is required.

— *from github.com/java-native-access/jna*

Hello World with JNA

```
#[no_mangle]
pub extern "C" fn hello() -> *const c_char {
    "Rusty Days!\0".as_ptr()
}
```

JNA - loading a dynamic library

```
internal interface LibGleanFFI : Library {
    companion object {
        internal var INSTANCE: LibGleanFFI =
            Native.load("glean_ffi", LibGleanFFI::class.java)
    }

    fun glean_initialize(cfg: FfiConfiguration): Byte

    fun glean_new_counter_metric(name: String, lifetime: Int): Long
}

// ...

LibGleanFFI.INSTANCE.glean_initialize(cfg)
```

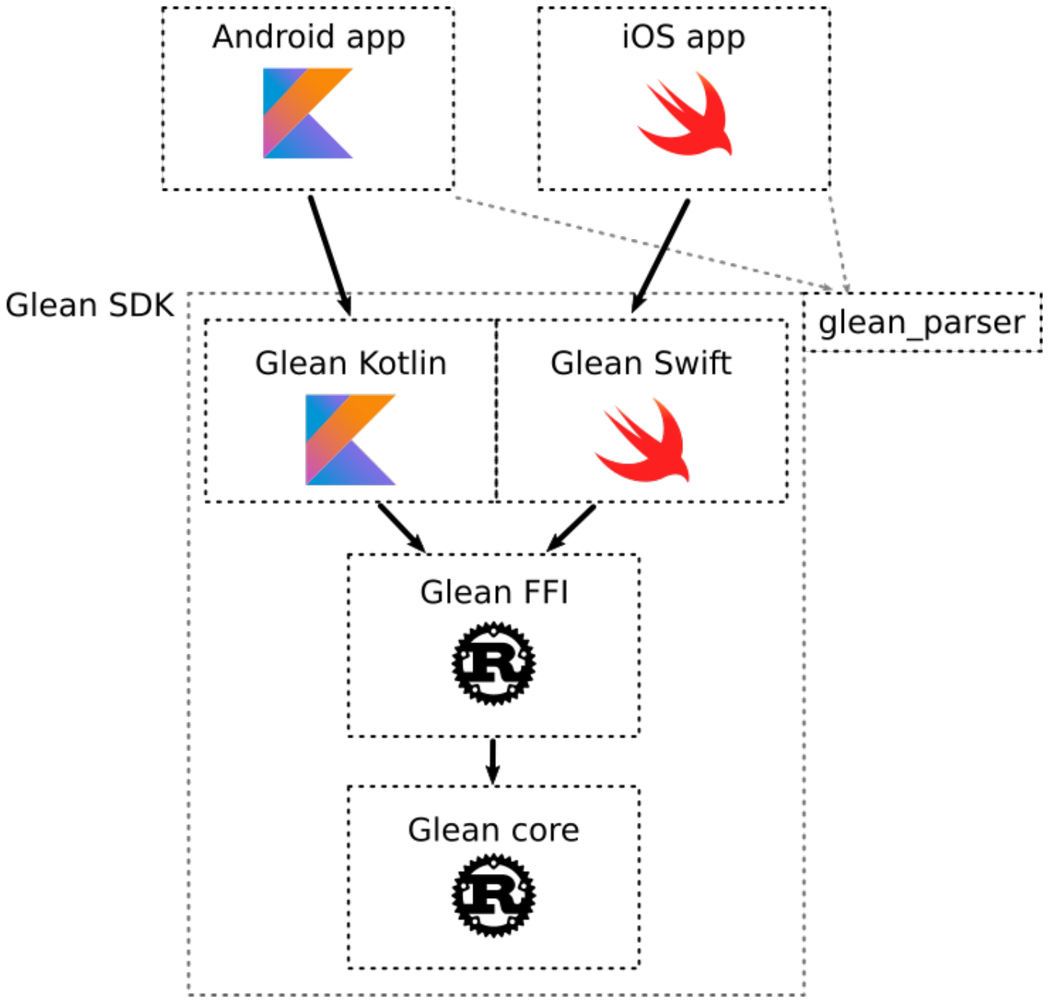
Building a Cargo project using Gradle[4]

```
apply plugin: 'org.mozilla.rust-android-gradle.rust-android'

cargo {
    module  = "../ffi"
    libname = "glean_ffi"
    targets = ["arm", "x86"]
}
```

[4] github.com/mozilla/rust-android-gradle

Other Glean implementations

- Swift - it speaks C!
- Python - similar to Swift, using cffi
- C# - similar to Kotlin
- Soon:
  - C++
  - JavaScript
  - Rust



Challenges of developing cross-platform Rust

CANCELED	iOS	pull/1052
FAILED	ci	pull/1052

Success	docs-spellcheck
Success	Rust tests - stable
Success	Rust tests - minimum version
Success	Python Windows x86_64 tests
Success	Python Windows i686 tests
Success	Python 3_8 tests minimum dependencies
Success	Python 3_8 tests
Success	Python 3_7 tests
Success	Python 3_6 tests minimum dependencies
Success	Python 3_6 tests
Success	Generate Rust documentation
Success	Generate Python documentation
Success	Generate Kotlin documentation
Success	docs-linkcheck
Success	CSharp tests
Failed	C tests
Failed	Android tests

Data types

Data types: Numbers

Rust	Kotlin	Swift
u8	Byte	UInt8
i32	Int	Int32
i64	Long	Int64
isize	IntegerType	Int

Data types: Bool in reality

```
1  # true
0  # false
```

```
1 bit
```

Data types: Bool in Rust

```
0000 0001  # true
0000 0000  # false
```

```
8 bit = 1 byte
```

Data types: Bool in Kotlin

```
0000 0000 0000 0000 0000 0000 0000 0001  # true
0000 0000 0000 0000 0000 0000 0000 0000  # false
```

32 bit = 4 byte

Data types: Strings

```
fun glean_submit_ping_by_name(
    ping_name: String,
    reason: String?
): Byte
```

Data types: Strings

```
// Rust
extern "C" fn glean_string_get(metric_id: i64) -> *mut c_char

// Kotlin
fun glean_string_get(metric_id: Long): Pointer?
```

Data types: String

```
fun Pointer.getAndConsumeRustString(): String {
    try {
        return this.getRustString()
    } finally {
        LibGleanFFI.INSTANCE.glean_str_free(this)
    }
}

fun Pointer.getRustString(): String {
    return this.getString(0, "utf8")
}
```

Data types: Plain ol' Enums

Kotlin

```
enum class TimeUnit {
    Microsecond,
    Millisecond,
    Second,
    Minute,
}
```

Rust

```
enum TimeUnit {
    Microsecond,
    Millisecond,
    Second,
    Minute,
}
```

Data types: Enums with data

Rust

```rust
#[repr(u8)]
pub enum FfiPingUploadTask {
    Upload {
        document_id: *mut c_char,
        body: ByteBuffer,
    },
    Wait,
    Done,
}
```

Tagged unions in C (through C bindgen)

```c
enum FfiPingUploadTask_Tag {
  FfiPingUploadTask_Upload,
  FfiPingUploadTask_Wait,
  FfiPingUploadTask_Done,
};
typedef uint8_t FfiPingUploadTask_Tag;

typedef struct {
  FfiPingUploadTask_Tag tag;
  char *document_id;
  ByteBuffer body;
} FfiPingUploadTask_Upload_Body;

typedef union {
  FfiPingUploadTask_Tag tag;
  FfiPingUploadTask_Upload_Body upload;
} FfiPingUploadTask;
```

Data types: Enums with data

Rust

```rust
#[repr(u8)]
pub enum FfiPingUploadTask {
    Upload {
        document_id: *mut c_char,
        body: ByteBuffer,
    },
    Wait,
    Done,
}
```

Tagged unions in Kotlin

```kotlin
enum class UploadTaskTag {
    Upload,
    Wait,
    Done
}

@Structure.FieldOrder("tag", "documentId", "body")
internal class UploadBody(
    val tag: Byte,
    val documentId: Pointer?,
    var body: RustBuffer,
) : Structure() { }

internal open class FfiPingUploadTask(
    var tag: Byte = UploadTaskTag.Done.ordinal.toByte(),
    var upload: UploadBody = UploadBody()
) : Union() {
}
```

Data types: Other rich data - JSON

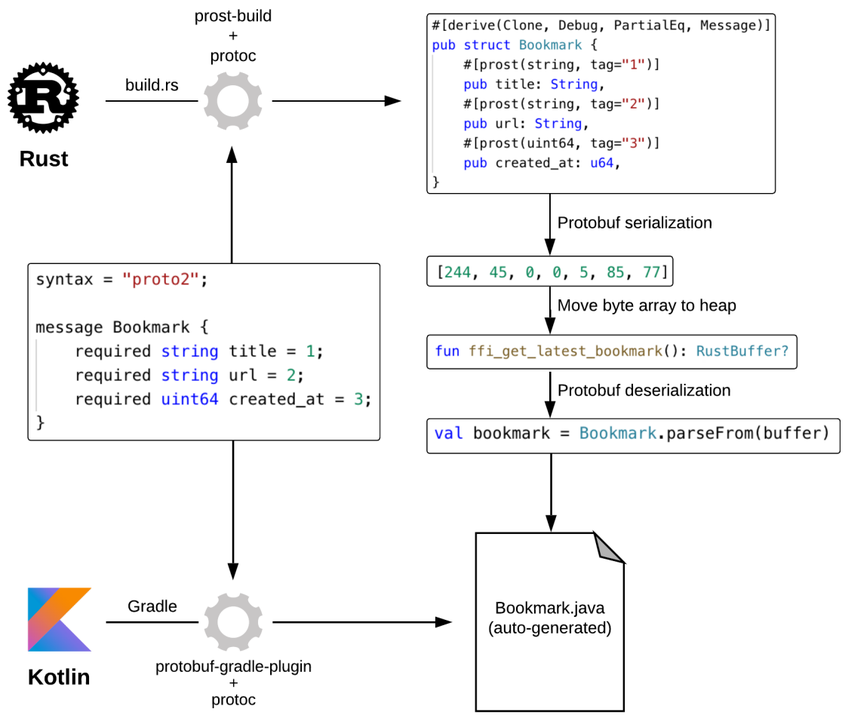
Rust

```
extern "C" fn glean_get_json(metric_id: i64)
    -> *mut c_char {
    let data = Glean.string_get(metric_id);
    let json = serde_json::to_string(&data);
    json.into_ffi_value()
}
```

Kotlin

```
// Declaration:
fun glean_get_json(metric_id: Long): Pointer?

// Usage:
val ptr = glean_get_json(handle)!!

jsonRes = JSONArray(ptr.getAndConsumeRustString())
return jsonRes.toList()
```

Data types: Other rich data - ProtoBuf[5]



[5] Crossing the Rust FFI frontier with Protocol Buffers

Optimizer: R8

- R8 minifies and optimizes the JVM bytecode
- It's buggy and might "over-optimize" JNA code

Optimizer: R8

proguard-consumer-rules.pro:

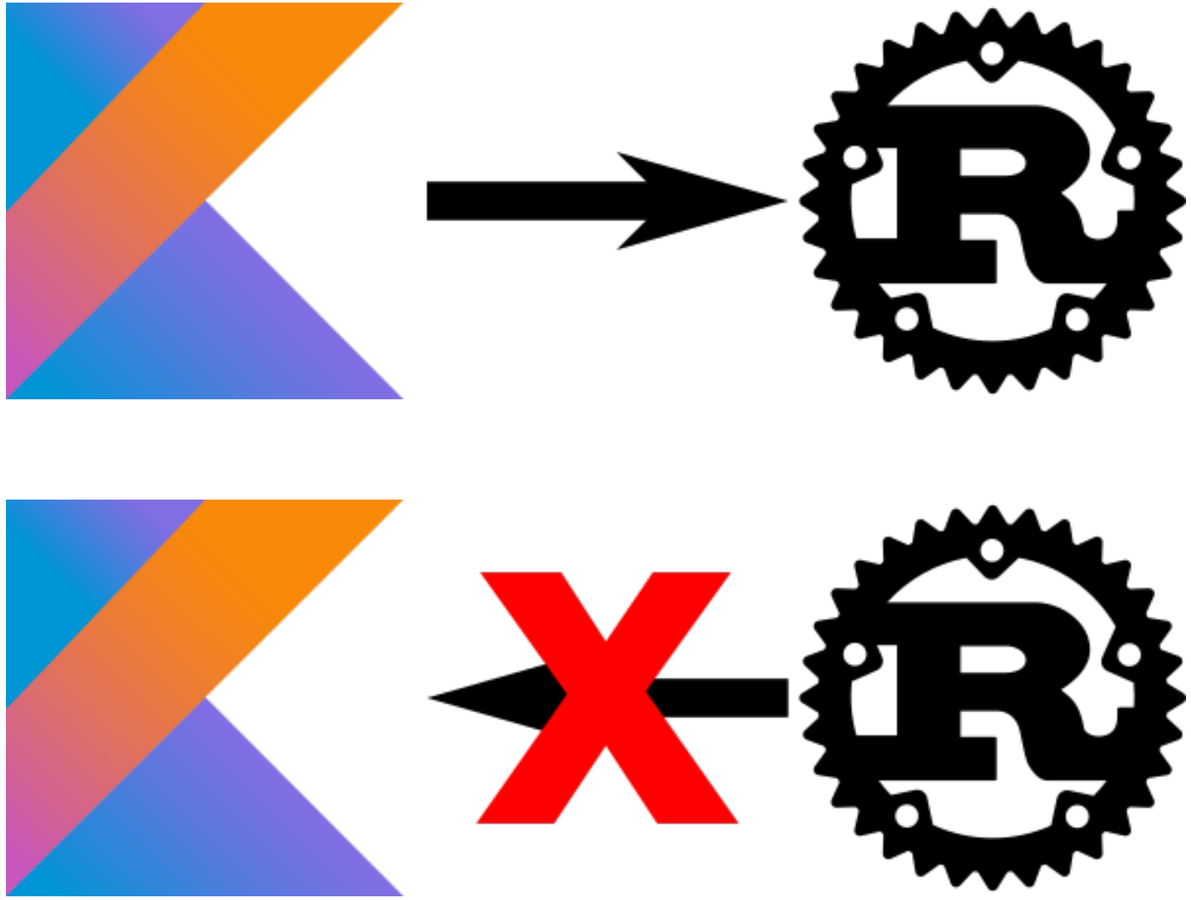
```
# JNA specific rules
-dontwarn java.awt.*
-keep class com.sun.jna.* { *; }
-keepclassmembers class * extends com.sun.jna.* { public *; }

# Glean specific rules
-keep class mozilla.telemetry.** { *; }
```

Extra libs

- `build.rs` exists, but everyone does it differently
- Build and link your C dependencies statically
- Consider precompiling them

What about the platform?

Glean goes on-direction only



Things that Kotlin does:

- Data storage path
- System & app information
- HTTP/network communication
- Time

The Future is Glean

telemetry for hu

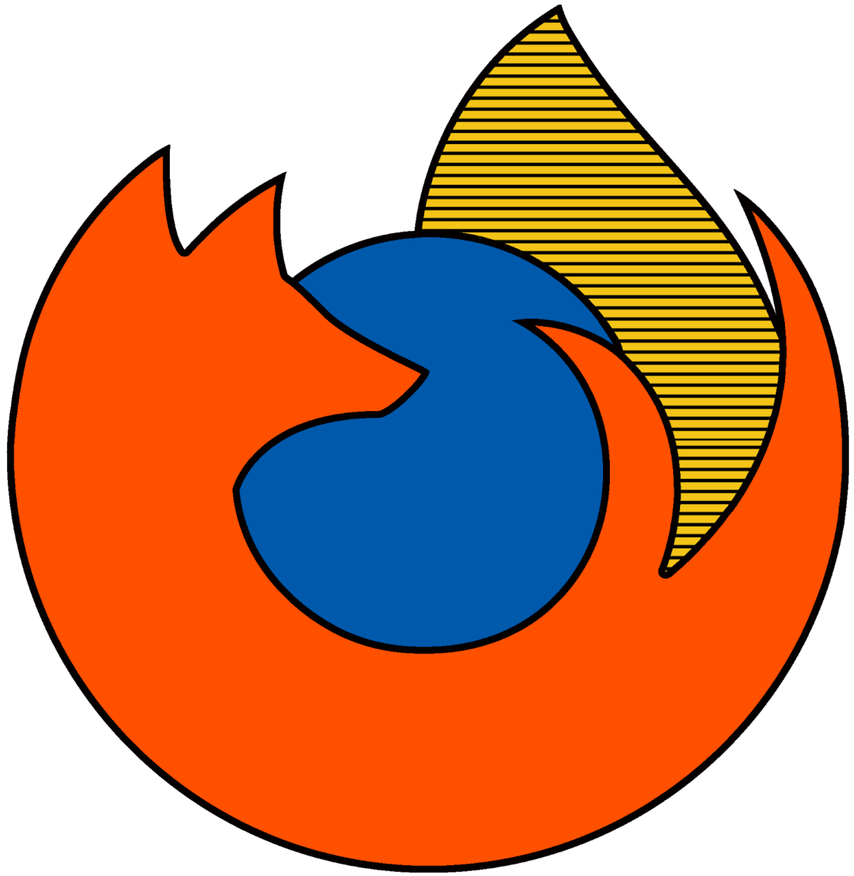
Future: Component Interface Definition

uniffi: Create boilerplate from IDL files[6]

```
interface Counter {
  constructor(string category, string name);
  void add(integer amount);
}
```

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

Future: **F**irefox **o**n **G**lean



Who else is using Rust to build cross-platform libraries, targetting mobile?

Async Rust, but using the platform?

Thanks to

the Telemetry team: Alessio, Bea, Chris, Travis, Mike and Georg.

the application-services team.

Links

- Slides: fnordig.de/talks/2020/rustydays/slides.pdf
- 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?