# UA Rust Conference 2024

July 27

online & offline

Learn. Develop. Discover.

All proceeds from the tickets will be donated to support **Ukraine**

near

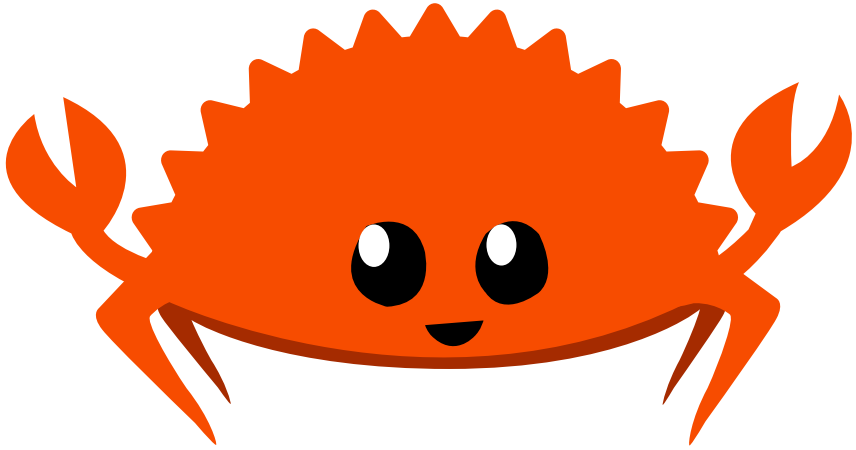
Campus community

kumeka team

BOHEMIA AUTOMATION

Out of the Box Systems

# A tale about implementing Storj Rust bindings



Ivan Fraixedes | @ifraixedes | ivan.fraixed.es

# A little bit about me

# Disclaimer

- Personal opinion, no the Storj's one
- Experienced Software Engineer, but mid-experience with Rust
- Rust bindings are NOT officially maintained by Storj

# What’s Storj?

Storj is a decentralized cloud storage

# What’s Storj?

## Storj is a decentralized cloud storage

Benefits:

- Data is stored in multiple nodes across the globe
- Secure by default
- Consistently fast
- Durability and availability without replication
- Possibility to use it as drop-in replacement for AWS S3

# What’s Storj?

## Storj is a decentralized cloud storage

Benefits:

- Data is stored in multiple nodes across the globe
- Secure by default
- Consistently fast
- Durability and availability without replication
- Possibility to use it as drop-in replacement for AWS S3

See a nice visual demo in how a file is uploaded and downloaded from Storj network at https://demo.storj.dev/

# Storj tech background

- Stoj is a Go shop
- libuplink is the client library to use Storj network through the native protocol
- libuplink is implemented in Go
- Libuplink C bindings

# Why Rust bindings?

To be able to use the Storj network native protocol

# Why Rust bindings?

To be able to use the Storj network native protocol

What does it mean?

# Why Rust bindings?

To be able to use the Storj network native protocol

What does it mean?

- Zero trust
- End to end connections

# Creating Rust bindings

# Creating Rust bindings

- The conventions are to generate Rust code that maps the C bindings through bindgen

# Creating Rust bindings

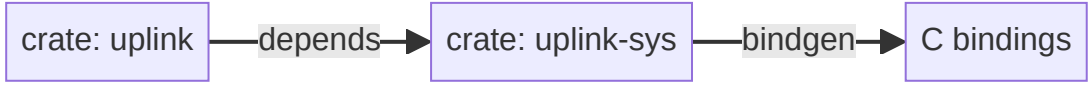
- The conventions are to generate Rust code that maps the C bindings through bindgen
- Create a first crate named `{name}-sys`
  - It calls bindgen referencing the C headers files & the library objects (binaries) for the platform that needs to build

# Creating Rust bindings

- The conventions are to generate Rust code that maps the C bindings through bindgen
- Create a first crate named `{name}-sys`
  - It calls bindgen referencing the C headers files & the library objects (binaries) for the platform that needs to build
- Create a second crate named `{name}` which has the `{name}-sys` as a dependency, and exposes the API safe and idiomatically

# Creating Rust bindings

- The conventions are to generate Rust code that maps the C bindings through bindgen
- Create a first crate named `{name}-sys`
  - It calls bindgen referencing the C headers files & the library objects (binaries) for the platform that needs to build
- Create a second crate named `{name}` which has the `{name}-sys` as a dependency, and exposes the API safe and idiomatically



# uplink-sys crate

# uplink-sys - peculiarities

- `uplink-c` is added as a git module
- `build.rs` compiles the `uplink-c`, so Go is needed

# uplink-sys - generated code

```c
typedef struct UplinkObject {
    char *key;
    bool is_prefix;
    UplinkSystemMetadata system;
    UplinkCustomMetadata custom;
} UplinkObject;
```

```rust
#[repr(C)]
#[derive(Debug, Copy, Clone)]
pub struct UplinkObject {
    pub key: *mut ::std::os::raw::c_char,
    pub is_prefix: bool,
    pub system: UplinkSystemMetadata,
    pub custom: UplinkCustomMetadata,
}
```

# uplink crate

# uplink - implementation

Best practices about documenting unsafe code

```rust
/// It returns the satellite node URL associated with this access grant.
pub fn satellite_address(&self) -> Result<&str, Error> {
    let strres;
    // SAFETY: we trust that the underlying c-binding is safe, nonetheless
    // we ensure strres is correct through the ensure method of the
    // implemented Ensurer trait.
    unsafe {
        strres = *ulksys::uplink_access_satellite_address(self.inner.access).ensure();
    }

    if let Some(e) = Error::new_uplink(strres.error) {
        return Err(e);
    }

    let addrres;
    // SAFETY: at this point we have already checked that strres.string is
    // NOT NULL
```

# uplink - implementation

Best practices about documenting unsafe code

```rust
/// It returns the satellite node URL associated with this access grant.
pub fn satellite_address(&self) -> Result<&str, Error> {
    let strres;
    // SAFETY: we trust that the underlying c-binding is safe, nonetheless
    // we ensure strres is correct through the ensure method of the
    // implemented Ensurer trait.
    unsafe {
        strres = *ulksys::uplink_access_satellite_address(self.inner.access).ensure();
    }

    if let Some(e) = Error::new_uplink(strres.error) {
        return Err(e);
    }

    let addrres;
    // SAFETY: at this point we have already checked that strres.string is
    // NOT NULL
```

# uplink - implementation

Best practices about documenting unsafe code

```rust
/// It returns the satellite node URL associated with this access grant.
pub fn satellite_address(&self) -> Result<&str, Error> {
    let strres;
    // SAFETY: we trust that the underlying c-binding is safe, nonetheless
    // we ensure strres is correct through the ensure method of the
    // implemented Ensurer trait.
    unsafe {
        strres = *ulksys::uplink_access_satellite_address(self.inner.access).ensure();
    }

    if let Some(e) = Error::new_uplink(strres.error) {
        return Err(e);
    }

    let addrres;
    // SAFETY: at this point we have already checked that strres.string is
    // NOT NULL
```

We ensured it with `#![deny(clippy::undocumented_unsafe_blocks)]`

# uplink - implementation

Specific Error type used consistently in the public API

```rust
/// The error type that this create use to wrap errors.
#[derive(Debug)]
pub enum Error {
    /// Identifies invalid arguments passed to a function or method.
    InvalidArguments(Args),
    /// Identifies a native error returned by the underlying Uplink C bindings
    /// library.
    Uplink(UplinkErrorDetails),
}
impl stderr::Error for Error {
    fn source(&self) -> Option<&(dyn stderr::Error + 'static)> {
        match self {
            Error::InvalidArguments { .. } => None,
            Error::Uplink { .. } => None,
        }
    }
}
impl fmt::Display for Error {
    fn fmt(&self, f: &mut fmt::Formatter) -> Result<(), fmt::Error> {
        match self {
            Error::InvalidArguments(args) => {
```

# uplink - implementation

Specific Error type used consistently in the public API

```rust
/// The error type that this create use to wrap errors.
#[derive(Debug)]
pub enum Error {
    /// Identifies invalid arguments passed to a function or method.
    InvalidArguments(Args),
    /// Identifies a native error returned by the underlying Uplink C bindings
    /// library.
    Uplink(UplinkErrorDetails),
}
impl stderr::Error for Error {
    fn source(&self) -> Option<&(dyn stderr::Error + 'static)> {
        match self {
            Error::InvalidArguments { .. } => None,
            Error::Uplink { .. } => None,
        }
    }
}
impl fmt::Display for Error {
    fn fmt(&self, f: &mut fmt::Formatter) -> Result<(), fmt::Error> {
        match self {
            Error::InvalidArguments(args) => {
```

# uplink - implementation

Go idioms and how they were mapped to C and Rust

# uplink - implementation

Go idioms and how they were mapped to C and Rust

```go
// When bucket already exists it returns a valid Bucket and ErrBucketExists.
func (project *Project) CreateBucket(ctx context.Context, bucket string) (created *Bucket, err error)
```

# uplink - implementation

Go idioms and how they were mapped to C and Rust

```go
// When bucket already exists it returns a valid Bucket and ErrBucketExists.
func (project *Project) CreateBucket(ctx context.Context, bucket string) (created *Bucket, err error)
```

# uplink - implementation

Go idioms and how they were mapped to C and Rust

```go
// When bucket already exists it returns a valid Bucket and ErrBucketExists.
func (project *Project) CreateBucket(ctx context.Context, bucket string) (created *Bucket, err error)
```

# uplink - implementation

Go idioms and how they were mapped to C and Rust

```go
// When bucket already exists it returns a valid Bucket and ErrBucketExists.
func (project *Project) CreateBucket(ctx context.Context, bucket string) (created *Bucket, err error)
```

# uplink - implementation

```rust
/// An interface for ensuring that an instance of type returned by the
/// underlying c-binding is correct in terms that it doesn't violate its own
/// rules.
/// For example a UplinkAccessResult struct has 2 fields which are 2 pointers,
/// one is the access and the other is an error, always one and only one can be
/// NULL.
trait Ensurer {
    /// Checks that the instance is correct according its own rules and it
    /// returns itself, otherwise it panics.
    fn ensure(&self) -> &Self;
}
```

# uplink - implementation

```rust
/// An interface for ensuring that an instance of type returned by the
/// underlying c-binding is correct in terms that it doesn't violate its own
/// rules.
/// For example a UplinkAccessResult struct has 2 fields which are 2 pointers,
/// one is the access and the other is an error, always one and only one can be
/// NULL.
trait Ensurer {
    /// Checks that the instance is correct according its own rules and it
    /// returns itself, otherwise it panics.
    fn ensure(&self) -> &Self;
}
```

# uplink - implementation

## Helpers

```rust
/// creates a CString from a function &str function argument and if there is an
/// error it returns an Error::InvalidArguments with the passed argument's
/// name.
pub fn cstring_from_str_fn_arg(arg_name: &str, arg_val: &str) -> Result<CString, Error> {
    match CString::new(arg_val) {
        Ok(cs) => Ok(cs),
        Err(e) => Err(Error::new_invalid_arguments(
            arg_name,
            &format!(
                "cannot contains null bytes (0 byte). Null byte found at {}",
                e.nul_position()
            ),
        )),
    }
}
```

# uplink - issues

# uplink - issue with docs.rs

# uplink - issue with docs.rs

- Modify the base Docker image

# uplink - issue with docs.rs

- Modify the base Docker image
- Adapt your crate's build with:
  - Detecting docs.rs build
  - Setting metadata for custom builds

# uplink - issue docs.rs

```rust
if env::var("DOCS_RS").is_err() {
    // Build uplink-c generates ....
    Command::new("make")
        .arg("build")
        .current_dir(&uplink_c_src)
        .status()
        .expect("Failed to run make command from build.rs.");
}

if env::var("DOCS_RS").is_ok() {
    // Use the precompiled uplink-c libraries for building the docs by docs.rs.
    Command::new("cp")
        .args(&[
            "-R",
            &PathBuf::from(".docs-rs").to_string_lossy(),
            &uplink_c_dir.join(".build").to_string_lossy(),
        ])
        .status()
        .expect("Failed to copy docs-rs precompiled uplink-c lib binaries");
```

# uplink - issue docs.rs

```rust
    Command::new("make")
        .arg("build")
        .current_dir(&uplink_c_src)
        .status()
        .expect("Failed to run make command from build.rs.");
}

if env::var("DOCS_RS").is_ok() {
    // Use the precompiled uplink-c libraries for building the docs by docs.rs.
    Command::new("cp")
        .args(&[
            "-R",
            &PathBuf::from(".docs-rs").to_string_lossy(),
            &uplink_c_dir.join(".build").to_string_lossy(),
        ])
        .status()
        .expect("Failed to copy docs-rs precompiled uplink-c lib binaries");
}
```

# uplink - issue docs.rs

```rust
    Command::new("make")
        .arg("build")
        .current_dir(&uplink_c_src)
        .status()
        .expect("Failed to run make command from build.rs.");
}

if env::var("DOCS_RS").is_ok() {
    // Use the precompiled uplink-c libraries for building the docs by docs.rs.
    Command::new("cp")
        .args(&[
            "-R",
            &PathBuf::from(".docs-rs").to_string_lossy(),
            &uplink_c_dir.join(".build").to_string_lossy(),
        ])
        .status()
        .expect("Failed to copy docs-rs precompiled uplink-c lib binaries");
}
```

```toml
[package.metadata.docs.rs]
default-target = "x86_64-unknown-linux-gnu"
targets = [] # Do not build the doc with any other target than the default.
```

# uplink - bugs

# Bug: `CStr` & `CString` misuse

```diff
/// Contains information about a specific bucket.
-pub struct Bucket<'a> {
+pub struct Bucket {
     /// Name of the bucket.
-    pub name: &'a str,
+    pub name: String,
     /// Unix Epoch time when the bucket was created.
     pub created_at: Duration,
 }

-impl<'a> Bucket<'a> {
+impl Bucket {
@@ -42,7 +42,8 @@ impl<'a> Bucket<'a> {
             // panic if they contain some and we return an internal error because we see it's a
             // limitation of Rust and C interoperability and consumers of this crate would have a
             // chance to deal with them appropriately.
-            name = CStr::from_ptr(uc_bucket.name).to_str().map_err(|err| {
+            let cs = CString::from(CStr::from_ptr(uc_bucket.name));
+            name = cs.into_string().map_err(|err| {
                 ulksys::uplink_free_bucket(uc_bucket_ptr);
```

# Bug: `CStr` & `CString` misuse

```diff
/// Contains information about a specific bucket.
-pub struct Bucket<'a> {
+pub struct Bucket {
     /// Name of the bucket.
-    pub name: &'a str,
+    pub name: String,
     /// Unix Epoch time when the bucket was created.
     pub created_at: Duration,
 }

-impl<'a> Bucket<'a> {
+impl Bucket {
@@ -42,7 +42,8 @@ impl<'a> Bucket<'a> {
             // panic if they contain some and we return an internal error because we see it's a
             // limitation of Rust and C interoperability and consumers of this crate would have a
             // chance to deal with them appropriately.
-            name = CStr::from_ptr(uc_bucket.name).to_str().map_err(|err| {
+            let cs = CString::from(CStr::from_ptr(uc_bucket.name));
+            name = cs.into_string().map_err(|err| {
                 ulksys::uplink_free_bucket(uc_bucket_ptr);
```

# Bug: `CStr` & `CString` misuse

```diff
/// Contains information about a specific bucket.
-pub struct Bucket<'a> {
+pub struct Bucket {
     /// Name of the bucket.
-    pub name: &'a str,
+    pub name: String,
     /// Unix Epoch time when the bucket was created.
     pub created_at: Duration,
 }

-impl<'a> Bucket<'a> {
+impl Bucket {
@@ -42,7 +42,8 @@ impl<'a> Bucket<'a> {
             // panic if they contain some and we return an internal error because we see it's a
             // limitation of Rust and C interoperability and consumers of this crate would have a
             // chance to deal with them appropriately.
-            name = CStr::from_ptr(uc_bucket.name).to_str().map_err(|err| {
+            let cs = CString::from(CStr::from_ptr(uc_bucket.name));
+            name = cs.into_string().map_err(|err| {
                 ulksys::uplink_free_bucket(uc_bucket_ptr);
```

# Bug: implementing `std::io::Read`

```diff
@@ -217,19 +217,41 @@ impl std::io::Read for Download {
     fn read(&mut self, buf: &mut [u8]) -> std::io::Result<usize> {
-        let bp = buf.as_mut_ptr();
-        let read_res = unsafe {
-            ulksys::uplink_download_read(self.inner.download, bp.cast(), buf.len() as u64)
-        };
-
-        if let Some(err) = Error::new_uplink(read_res.error) {
-            use std::io::{Error as IOErr, ErrorKind};
-            return Err(IOErr::new(ErrorKind::Other, err));
+        for _ in 1..3 {
+            let bp = buf.as_mut_ptr();
+            let read_res = unsafe {
+                ulksys::uplink_download_read(self.inner.download, bp.cast(), buf.len() as u64)
+            };
+
+            if let Some(err) = Error::new_uplink(read_res.error) {
+                if let Error::Uplink(error::Uplink::Unknown(_)) = err {
+                    return Ok(read_res.bytes_read as usize);
+                }
+
+                use std::io::{Error as IOErr, ErrorKind};
```

# Bug: implementing `std::io::Read`

```
12  +            let bp = buf.as_mut_ptr();
13  +            let read_res = unsafe {
14  +                ulksys::uplink_download_read(self.inner.download, bp.cast(), buf.len() as u64)
15  +            };
16  +
17  +            if let Some(err) = Error::new_uplink(read_res.error) {
18  +                if let Error::Uplink(error::Uplink::Unknown(_)) = err {
19  +                    return Ok(read_res.bytes_read as usize);
20  +                }
21  +
22  +                use std::io::{Error as IOErr, ErrorKind};
23  +                return Err(IOErr::new(ErrorKind::Other, err));
24  +            }
25  +
26  +            if read_res.bytes_read ≠ 0 {
27  +                return Ok(read_res.bytes_read as usize);
28  +            }
29          }
30
31  -       Ok(read_res.bytes_read as usize)
32  +       Ok(0)
33      }
```

# Bug: implementing `std::io::Read`

```
10  -           return Err(IOErr::new(ErrorKind::Other, err));
11  +       for _ in 1..3 {
12  +           let bp = buf.as_mut_ptr();
13  +           let read_res = unsafe {
14  +               ulksys::uplink_download_read(self.inner.download, bp.cast(), buf.len() as u64)
15  +           };
16  +
17  +           if let Some(err) = Error::new_uplink(read_res.error) {
18  +               if let Error::Uplink(error::Uplink::Unknown(_)) = err {
19  +                   return Ok(read_res.bytes_read as usize);
20  +               }
21  +
22  +               use std::io::{Error as IOErr, ErrorKind};
23  +               return Err(IOErr::new(ErrorKind::Other, err));
24  +           }
25  +
26  +           if read_res.bytes_read ≠ 0 {
27  +               return Ok(read_res.bytes_read as usize);
28  +           }
29          }
30
31  -       Ok(read_res.bytes_read as usize)
32  +       Ok(0)
```

# Project highlights

github.com/storj-thirdparty/uplink-rust/

# Project highlights

github.com/storj-thirdparty/uplink-rust/

- Completed and up to date

# Project highlights

github.com/storj-thirdparty/uplink-rust/

- Completed and up to date
- Tests. Unit and integration tests

# Project highlights

github.com/storj-thirdparty/uplink-rust/

- Completed and up to date
- Tests. Unit and integration tests
- CI (Github Actions)

# Project highlights

github.com/storj-thirdparty/uplink-rust/

- Completed and up to date
- Tests. Unit and integration tests
- CI (Github Actions)
- Good commits

# Project highlights

github.com/storj-thirdparty/uplink-rust/

- Completed and up to date
- Tests. Unit and integration tests
- CI (Github Actions)
- Good commits
- Fully and well documented ( `#![deny(missing_docs)]` )

# Project highlights

github.com/storj-thirdparty/uplink-rust/

- Completed and up to date
- Tests. Unit and integration tests
- CI (Github Actions)
- Good commits
- Fully and well documented ( `#![deny(missing_docs)]` )

**There is room for improvement, feel free to contribute if you wish**

# Questions?

**Gratitude**

- Storj: Storj logo and their innovation time program.
- Hello crustaceans!: Ferris logo under the public domain.
- Egon Elbre: Gophers logo under public domain.
- Ukrainian Rust Community: To accept my talk & the support to improve it

# UA Rust Conference 2024

July 27

online & offline

Learn. Develop. Discover.

All proceeds from the tickets will be donated to support **Ukraine**

near

Campus community

kumeka team

BOHEMIA AUTOMATION

Out of the Box Systems