Calling JIT compiled scripts

Under the hood of Roto's integration with Rust

Feb 01 2026, FOSDEM

Terts Diepraam (he/him), NLnet Labs

Quick disclaimer

NLnet Labs ≠ NLnet

We can’t fund your projects (sorry!)

NLnet Labs

Open Source

DNS and routing

Non-profit organization

Been around for 26 years!

E.g. NSD, Unbound, Routinator, Rotonda

Rotonda

BGP route collector

Written in Rust

Has to filter *a lot* of data

Requires a **fast** scripting language

Me at FOSDEM 2024

Master’s thesis in Programming Languages

Me at FOSDEM 2024

Master’s thesis in Programming Languages

Looking for a job

Me at FOSDEM 2024

Master’s thesis in Programming Languages

~~Looking for a job~~

Software Engineer at NLnet Labs

Me at FOSDEM 2024

Master's thesis in Programming Languages

~~Looking for a job~~

Software Engineer at NLnet Labs

Organizer of RustWeek

Chapter 1

Rotonda

Chapter 1

Roto~~nda~~

In a nutshell

Statically typed

Integrates tightly with Rust

JIT compiled to machine code!

No interpreter, no bytecode

Example

```
# script.roto
fn foo(x: f64) -> f64 {
    print(f"Got value: {x}");
    (2.0 * x).pow(0.5)
}
```

Example

```
# script.roto
fn foo(x: f64) -> f64 {
    print(f"Got value: {x}");
    (2.0 * x).pow(0.5)
}
```

```
// main.rs
let rt = roto::Runtime::new();
let mut pkg = rt.compile("script.roto")?;
let f = pkg.get_function::<fn(f64) -> f64>("foo")?;
let y = f.call(10.0);
```

Chapter 2
Under the Hood

Cranelift

Roto compiles to Cranelift IR

Cranelift then compiles that to machine code!

The unsafest `unsafe`

Cranelift gives us just a pointer and a buffer of code.

The unsafest `unsafe`

Cranelift gives us just a pointer and a buffer of code.

```rust
// SAFETY: ???
let func_ptr = unsafe {
    mem::transmute::<*const u8, Self::RotoFn>(ptr)
};
```

where `Self::RotoFn` is an `extern "C" fn`

The unsafest `unsafe`

Cranelift gives us just a pointer and a buffer of code.

```rust
// SAFETY: ???
let func_ptr = unsafe {
    mem::transmute::<*const u8, Self::RotoFn>(ptr)
};
```

where `Self::RotoFn` is an `extern "C" fn`

🔥🐕🔥 *“This is fine”*

Getting the signature

Each argument and return type needs to map to a Roto type

So we can check against the signature in the script

Sending Rust to Roto

$$\begin{tikzcd}
\texttt{bool} \arrow[r] & \texttt{bool} \\
\texttt{u8} \arrow[r] & \texttt{u8} \\
\vdots & \vdots
\end{tikzcd}$$

Type layout

We can only deal with values with a known representation.

Otherwise we don’t know what code to generate!

Type layout

We can only deal with values with a known representation.

Otherwise we don’t know what code to generate!

```
std::mem::{size_of, align_of}
```

Type layout

We can only deal with values with a known representation.

Otherwise we don’t know what code to generate!

`std::mem::{size_of, align_of}`

Primitives are well-defined!

Type layout

We can only deal with values with a known representation.

Otherwise we don't know what code to generate!

`std::mem::{size_of, align_of}`

Primitives are well-defined!

But `Option<T>` ?

Type layout

> *The **only data layout guarantees** made by this representation are those required for soundness. They are:*
>
> - *The fields are properly aligned.*
> - *The fields do not overlap.*
> - *The alignment of the type is at least the maximum alignment of its fields.*
>
> — *Rust Reference*

A normal `Option<T>` could be anything! 😔

Type layout: `#[repr(C)`

```rust
#[repr(C)]
enum RotoOption<T> {
    Some(T),
    None
}
```

So we have to transform to `C` representation!

Sending Rust to Roto

`bool`	$\longrightarrow$	`bool`
`u8`	$\longrightarrow$	`u8`
$\vdots$		$\vdots$
`Option<T>`	$\longrightarrow$	`RotoOption<T>`

Chapter 3 Registration

Builtins

Strings

Booleans

Integers

Floats

…

But not everything!

DateTime?

```rust
fn fast_forward(dt: DateTime) -> DateTime {
    dt.add_days(7)
}
```

Register it!

```rust
use jiff::Zoned;
use roto::{Runtime, Val, library};

let lib = library! {
    #[clone] type DateTime = Val<Zoned>;
};

let rt = Runtime::from_lib(lib)?;
```

`Val<T>` means "custom type"

`#[clone]` means "`Clone` but not `Copy`"

DateTime!

```
# script.roto
fn fast_forward(dt: DateTime) -> DateTime {
    dt # TODO: add some days
}
```

```
// main.rs
type F = fn(Val<Zoned>) -> Val<Zoned>;

let f = pkg.get_function::<F>("fast_forward")?;

let now = Val(Zoned::now());
let out = f.call(now);
```

Sending Rust to Roto

`bool`	⟶	`bool`
`u8`	⟶	`u8`
⋮		⋮
`Option<T>`	⟶	`RotoOption<T>`
`Val<T>`	⟶	`*const u8`

Time to add functionality!

```rust
use jiff::{ToSpan, Zoned};
use roto::{Runtime, Val, library};

let lib = library! {
    #[clone] type DateTime = Val<Zoned>;

    impl Val<Zoned> {
        fn add_days(self, num_days: i64) -> Self {
            Val(self.0 + num_days.days())
        }
    }
};

let rt = Runtime::from_lib(lib)?;
```

Function registration

Very similar to `RotoFunc`

Reuse the `Value` trait!

Do the opposite transformations

Chapter 4

Generics

Lists

We should have `List`

And we should implement it in Rust!

```
let lib = library! {
    #[clone] type List<T> = Vec<T>;
};
```

Lists

We should have `List`

And we should implement it in Rust!

```
let lib = library! {
    #[clone] type List<T> = Vec<T>;
};
```

Nope!

But...

Generic types don't exist after compilation

No `TypeId`, fixed layout, function pointers

End of the road?

But what if…

But what if… we fake generics?

Fake generics!

We can register *type-erased* data structures

`List<T>` ≈ `ErasedList` + vtable of `T`

Pretend there’s a type parameter

Sending Rust to Roto

`bool`	⟶	`bool`
`u8`	⟶	`u8`
⋮		⋮
`Option<T>`	⟶	`RotoOption<T>`
`Val<T>`	⟶	`*const u8`
`List<T>`	⟶	`*const ErasedList`

Lists in action

```
# script.roto
fn foo(x: List[u8]) -> List[u8] {
  x.push(42);
  x
}
```

```
// main.rs
let f = pkg
  .get_function::<fn(List<u8>) -> List<u8>>("foo")?;
let y = f.call(List::new());
```

Chapter 5

Epilogue

General strategies

Don’t be too clever

Test test test!

Run Valgrind & MIRI

More features coming!

Maps / Dictionaries

Accessing Rust `struct` fields

Matching on Rust `enums`

First class functions

and more!

Join us for RustWeek 2026!

RustWeek

by RustNL

May 18-23, 2026 – Utrecht, The Netherlands

See rustweek.org

Join us for RustWeek 2026!

DISCOUNT CODE: FOSDEM26

26 – Utrecht, The Netherlands

See rustweek.org

Links

More about Roto

- github.com/NLnetLabs/roto
- roto.docs.nlnetlabs.nl

Find me online

- terts.dev
- terts@nlnetlabs.nl
- @mastodon.online@tertsdiepraam

Feel free to come up and talk to me!

Slides made with Typst.

No GenAI was used.

Slides, recording & links:

https://terts.dev/talks/roto-fosdem26