How to Exchange Rot for Rust
Exchange

What we’re working on

Support for Microsoft Exchange Web Services mail protocol

- First Rust component for Thunderbird
- First mail protocol to be added in Thunderbird’s lifetime
No one knew how to support a new protocol
Rot
Decaying architecture, unmaintained code

**A brief history of Thunderbird**

- Like Firefox, grew out of Netscape Communicator
  - 0.1 released in July 2003, 1.0 released in December 2004
- Mozilla divested, transferred ownership to community in 2012
- Maintained by the community until rejoining Mozilla Foundation in 2017
Rot

Decaying architecture, unmaintained code

What does that mean for the project?

- Long period of ad hoc changes and fixes without overarching architectural vision
- Loss of institutional knowledge
- No major architectural maintenance in over 20 years
- Decaying C++, not using modern standards
A significant challenge, but a significant opportunity
Rust
Rust

Why we chose it

Benefits to a small team

• All the usual reasons
  • Memory safety
  • Performance
  • Modularity and ecosystem
Rust

Why we chose it

Firefox

- Thunderbird is built on top of Gecko
- Build system and CI tooling already in place
- Integrated into XPCOM, the cross-language interface
Rust

Why we chose it

Looking ahead

• “Permission” to reconsider architecture
• Breaks reliance on old, delicate code paths
• Documentation tooling
Rust

The problems we encountered

Large extant codebase

• APIs and designs which don’t match Rust idioms
• Lots of existing features and capabilities which don’t integrate with the ecosystem
• Widespread idiosyncratic async patterns
Rust

The problems we encountered

XPCOM + Rust developer experience

• Thunderbird much more reliant on XPCOM than Firefox
  • Part of our aging architecture
  • Many interfaces, large surface areas, lots of inheritance
• Bindings built around C++ ABI for performance
• Limitations in Rust tooling around including generated bindings
Rust

The problems we encountered

The build system

- Firefox has a C++ entrypoint
  - No single point of entry for Rust code
  - All crates into a single workspace to avoid duplication
- Thunderbird built as a subtree of Firefox
  - cargo doesn’t like that
- Solution (kinda): script to merge dependencies and vendor
We can use Rust in Thunderbird! 🎉

What do we do with it now?
What are we trying to achieve?

- Support Microsoft Exchange
  - EWS (Exchange Web Services)
- EWS $\Rightarrow$ XML (SOAP) over HTTP
- More code infrastructure required:
  - Send HTTP requests through Necko
  - (De)Serialize XML data with scale
Sending HTTP requests

Interacting with XPCOM

- Cross-Platform Component Object Model
- Inter-components interaction
- Platform-neutral interfaces (XPIDL)
- Crossing language boundaries
- Let’s use it to interact with Necko!
Sending HTTP requests
Sending HTTP requests

Step 1: Support async/await

- New internal crate (`xpcom_async`)
- XPCOM async ⇒ Rust’s native async syntax
- Custom stream listener:
  - Buffers incoming data
  - Wakes a `std::task::Waker` when the request finishes
- Wrapped in `XpComFuture`:
  - Triggers XPCOM’s async
  - Implements `std::future::Future`
Step 2: Idiomatic HTTP

- Another new internal crate (`moz_http`)
- Native async interface with `xpcom_async`
- Rust-idiomatic, request-like HTTP client
- Creates and configures XPCOM objects, wrapped into `XpComFuture`
- Nice error handling
unsafe { demo() }
Handling XML content

Initial exploration

- Issues with most existing XML crates:
  - Handling namespaces and attributes
  - Very boilerplatey

- Fine for deserialization, not serialization
  - Need namespaces and attributes in requests
  - Loads of data structures and operations in EWS \(\Rightarrow\) low boilerplate
Handling XML content

Serializing XML

- External crate (`xml_struct`)
- Code generation using Rust’s procedural macros
- Dynamic trait implementations at compile time (`derive`)
- Built on top of `quick-xml`
- Low-boilerplate approach
unsafe { demo() }
What’s next?

- Implement the damn thing!
  - Implement protocol support for EWS in Rust
  - Hook this support to the Thunderbird UI
- Bonus points: generalize the `xml_struct` crate if there is enough interest
Thank you!