Rust WG Home

Rust WG	DRAFT
Mailing List	https://lists.aswf.io/g/wg-rust
Slack channel	https://slack.aswf.io #rust
Meeting Information	Working Group meets bi-weekly on Thursdays at 1pm Pacific time. Zoom: https://zoom.us/j/97892963693
TAC Member Sponsor	
Chairperson(s)	Scott Wilson, Robin Rowe

See also the vfx-rs/organization GitHub project for up-to-date group initiatives, policies, and project statuses.

Purpose

The Rust bindings working group is dedicated to creating a foundation for C and Rust bindings for C++ libraries used by the media and entertainment industry.

Goals

Goals are the distinct outcomes that is to be anticipated from the working group, serving as a method for validating activities.

- To collaborate with ASWF communities to create (or facilitate the creation of) C and Rust bindings which are consistent across projects.
- Create tooling to help build the C and Rust bindings.
- Define a set of conventions and rules to create a consistent and ergonomic C and Rust interface for the wrapped libraries.
- To have the Rust bindings easily accessible through Rust's crates.io repository, and to give ownership of those bindings to the ASWF.

Non-Goals

Non-goals are things that the working group are intentionally choosing not to do, the things not expected to change due to the working groups efforts, or the subject matter areas that the working groups doesn't want to address at this time.

- Re-implementing current C++ libraries in Rust.

Deliverables

- Bind Imath and OpenEXR with C and Rust bindings
- Create a framework for binding the other libraries

Meeting notes

Meeting notes, recordings, and any presentations made during WG meetings are available here.

Meeting Notes: ASWF Rust WG Meeting July 13th, 2023

By Robin Rowe. Discussion of how Rust may be used for pipeline glue code. Rust, as an emerging language, is not in use yet for building core games on commercial game platforms.

Rust is being used in a supporting role in studio development (games, vfx, animation) that would otherwise be in C#, Bash or Python. C# seems on a path like C++, with too much baggage and complexity for many users. Bash is not an elegant programming language to use, but is great that it is everywhere and requires no build frameworks because it is interpreted. For studio pipelines, Python is often chosen as a better Bash, with core code written in C++ for performance. Python offers a nice debugging environment in Visual Studio Code and more modern code constructs. However, Python is slow. While the performance hit may be inconsequential to typical glue code, Python, or any interpreted language, can be expensive to test and to CI/CD.

Rust is finding application in studio pipeline programs that process a lot of text, such as diagnostics. Traditionally, this would be handled in Perl, valued for its regex prowess. Studios have mostly abandoned Perl in favor of Python. Being compiled, Rust offers a more performant alternative for text processing that Perl or Python.

Rust code may run 10x faster than interpreted languages like Bash and Python. Rust has Cargo as its build system. Much simpler than using make, cmake or ugh, autotools. Thanks to Cargo, that Rust is compiled instead of interpreted isn't a productivity hit. Unlike C/C++ where for small programs it can seem too much work to configure building a small program. However, some C/C++ programmers use the open source Cmaker program on gitlab that quickly auto-generates Cmake files something like using Cargo.

Rust has destructors, in the form of drop(), but unlike C++ has no constructors. Instead, Rust requires all data be initialized. Instead of using C++ operator! to check the health of an object, Rust uses its ? operator.

Rust has implicit return. When the last line of a Rust function ends without a semi-colon it is a return value. This looks a little weird to programmers accustomed to other languages that call 'return'.

These links were shared during the meeting:

- https://crates.io/crates/indicatif
- https://docs.rs/tui/latest/tui/
- **tui - Rust**
- tui is a library used to build rich terminal users interfaces and dashboards.
- https://doc.rust-lang.org/std/ops/trait.Drop.html
- **Drop in std::ops - Rust**
- Custom code within the destructor.
- https://doc.rust-lang.org/book/ch09-02-recoverable-errors-with-result.html
- https://doc.rust-lang.org/rust-by-example/std_misc/file/read_lines.html
- **read lines - Rust By Example**
- Rust by Example (RBE) is a collection of runnable examples that illustrate various Rust concepts and standard libraries.

Note, the next ASWF Rust WG meeting is on July 27th, 2023.

Recent space activity

Recently Updated

As you and your team create content this area will fill up and display the latest updates.

Space contributors

No contributors found for: authors on selected page(s)