

READY FOR RUST

Erik Dörnenburg | ThoughtWorks | @erikdoe

[Overview](#)[Key Results](#)[Developer Profile](#)**Technology**[1. Most Popular Technologies](#)**[2. Most Loved, Dreaded, and Wanted](#)**[3. Development Environments and Tools](#)[4. Blockchain in the Real World](#)[5. Top Paying Technologies](#)[6. Correlated Technologies](#)[Work](#)[Community](#)[Methodology](#)[Back to top](#)

Take control of your job search.

Stack Overflow Jobs puts developers first, so recruiters aren't at the top of the job listings.

[Browse jobs](#)

Find your next developer.

Source, attract and recruit developers on the platform they trust most.

[Learn more](#)

Your private version of Stack Overflow.

Create a secure space for your engineering team to ask and

Most Loved, Dreaded, and Wanted Languages

[Loved](#) [Dreaded](#) [Wanted](#)

[Overview](#)[Key Results](#)[Developer Profile](#)**Technology****I. Most Popular Technologies**[II. Most Loved, Dreaded, and Wanted](#)[III. Development Environments and Tools](#)[IV. Blockchain in the Real World](#)[V. Top Paying Technologies](#)[VI. Correlated Technologies](#)[Work](#)[Community](#)[Methodology](#)[Back to top](#)

Take control of your job search.

Stack Overflow Jobs puts developers first, so recruiter spam or fake job listings.

[Browse jobs](#)

Find your next developer.

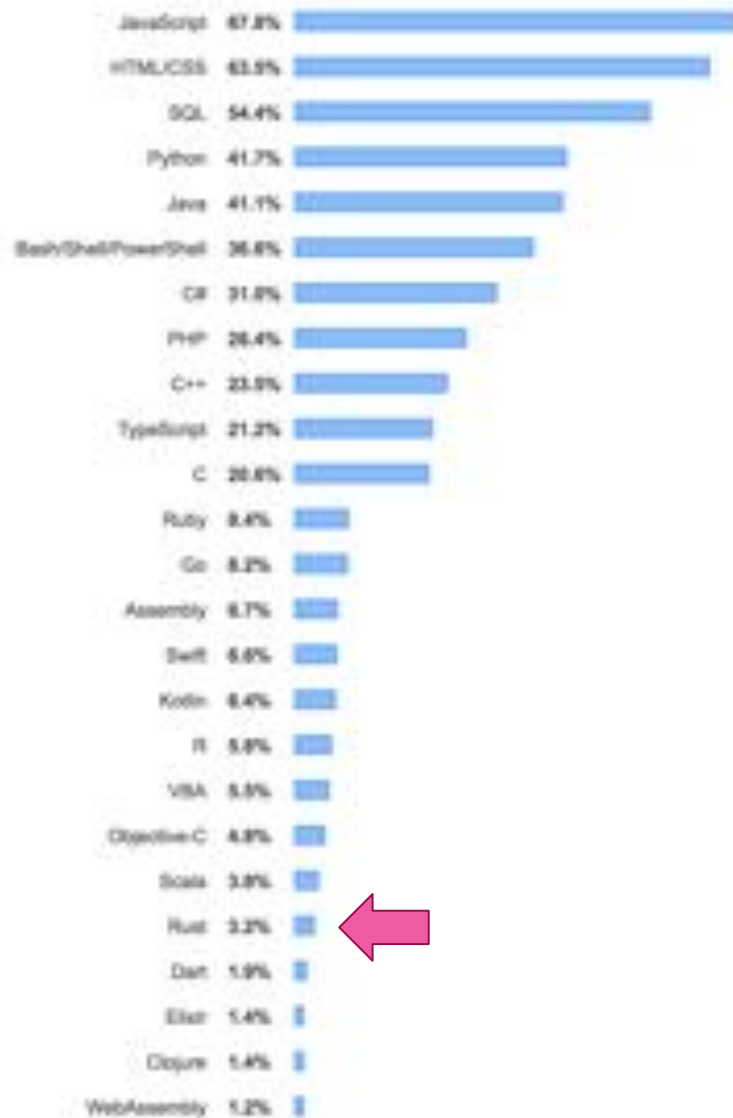
Source, attract and recruit developers on the platform they trust most.

[Learn more](#)

Your private version of Stack Overflow.

Create a secure space for your engineering team to ask and answer questions.

Programming, Scripting, and Markup Languages

[All Respondents](#)[Professional Developers](#)



Google

moz://a



Swift



**Apply the security updates to your OS X Macs
to help protect against known and future
issues. Security updates 2014-001 through
2014-004**

These updates address known and future issues with
OS X. For more information, see the following links:

How to update security updates

- [How to update security updates](#)
- [How to update security updates](#)
- [How to update security updates](#)
- [How to update security updates](#)

**How to update the OS X Security updates 2014-001 through
2014-004. Security updates 2014-001 through
2014-004**

- [How to update the OS X Security updates 2014-001 through 2014-004](#)
- [How to update the OS X Security updates 2014-001 through 2014-004](#)
- [How to update the OS X Security updates 2014-001 through 2014-004](#)
- [How to update the OS X Security updates 2014-001 through 2014-004](#)

[Blurred text with four pink arrows pointing to specific lines]

1.1 **Introduction**

The purpose of this report is to provide a comprehensive overview of the current state of the market for [Product/Service]. This report will analyze the market's growth, key players, and emerging trends. The findings will be used to inform strategic decisions and identify opportunities for [Company Name].

The report is structured as follows:

- 1.1 Introduction
- 1.2 Market Overview
- 1.3 Key Players
- 1.4 Emerging Trends
- 1.5 Conclusion

1.2 **Market Overview**

The market for [Product/Service] has experienced significant growth over the past five years, driven by increasing demand and technological advancements. The market is expected to continue to grow, with a projected CAGR of [X%] from 2023 to 2028.

Key factors driving market growth include:

- Increasing demand for [Product/Service]
- Technological advancements in [Product/Service]
- Government support and incentives

The market is highly competitive, with several key players vying for market share. The top three players are [Company A], [Company B], and [Company C].

1.3 **Key Players**

The following table provides a summary of the key players in the market:

Company Name	Market Share (%)	Key Products/Services
[Company A]	[X%]	[Product/Service 1], [Product/Service 2]
[Company B]	[X%]	[Product/Service 1], [Product/Service 2]
[Company C]	[X%]	[Product/Service 1], [Product/Service 2]

1.4 **Emerging Trends**

Several emerging trends are expected to shape the market in the coming years:

- Increased focus on sustainability and environmental impact
- Integration of artificial intelligence and machine learning
- Expansion into new geographic markets

1.5 **Conclusion**

The market for [Product/Service] is highly dynamic and offers significant opportunities for growth. [Company Name] should focus on [Key Strategy] to maintain its competitive edge and capitalize on emerging trends.





NEWS

C++ Memory Bugs Prompt Microsoft to Eye Rust Instead

By David Ramel ■ 07/18/2019

Microsoft is eyeing the Rust programming language as a safer replacement of C/C++ code after discovering just how many security problems are caused by memory corruption bugs.



That news comes in a new blog post by the Microsoft Security Response Center (MSRC), which in triaging every reported Microsoft vulnerability since 2004 found that "one astonishing fact sticks out."

That astonishing fact? "The majority of vulnerabilities fixed and with a CVE

.NET Insight

Sign up for our newsletter.

Email Address:

Click to Select One



SUBMIT

[I agree to this site's Privacy Policy.](#)

Most Popular Articles

What Are gRPC Web Services and When Should I Use Them?

How to Integrate Blazor Components



“The majority of vulnerabilities fixed and with a CVE assigned are caused by developers inadvertently inserting memory corruption bugs into their C and C++ code”

– *Microsoft Security Response Center: A proactive approach to more secure code (July 2019)*





WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Content page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Language 🌐
Deutsch
Español
Français
Galego
Italiano
日本語
한국어
Қазақша
Српски
ไทย

🔍 1000/1000

Article Talk

Read Edit View history Search Wikipedia



ISO 26262

From Wikipedia, the free encyclopedia



This article needs to be **updated**. Please update this article to reflect recent events or newly available information.

Last update: 2011 (November 2016)

ISO 26262, titled "Road vehicles – Functional safety", is an international standard for functional safety of electrical and/or electronic systems in production automobiles defined by the International Organization for Standardization (ISO) in 2011.

Contents [hide]

- Overview of Part 1
- Parts of ISO 26262
 - Part 1: Vocabulary
 - Part 2: Management of functional safety
 - Parts 3-7: Safety Life Cycle
 - Part 8: Supporting Processes
 - Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis
 - ASIL Assessment Overview
 - ASIL Assessment Process
- See also
- References
- External links

Overview of Part 1 [[edit](#)]

Functional safety features form an integral part of each automotive product development phase, ranging from the specification, to design, implementation, integration, verification, validation, and production release. The standard ISO 26262 is an adaptation of the Functional Safety standard IEC 61508 for Automotive Electric/Electronic Systems. ISO 26262 defines functional safety for automotive equipment applicable throughout the lifecycle of all automotive electronic and electrical safety-related systems.

The first edition, published on 11 November 2011, is intended to be applied to electrical and/or electronic systems installed in "series production passenger cars" with a maximum gross weight of 3500 kg. It aims to address possible hazards caused by the malfunctioning behaviour of electronic and electrical systems.

Although entitled "Road vehicles – Functional safety" the standard relates to the functional safety of Electrical and Electronic systems as well as that of systems as a whole or of their mechanical subsystems.

Like its parent standard, IEC 61508, ISO 26262 is a risk-based safety standard, where the risk of hazardous operational situations is qualitatively assessed and safety measures are defined to avoid or control systematic failures and to detect or control random hardware failures, or mitigate their effects.

Goals of ISO 26262:

- Provides an automotive safety lifecycle (management, development, production, operation, service, decommissioning) and supports tailoring the necessary activities during these lifecycle phases.
- Covers functional safety aspects of the entire development process (including such activities as requirements specification, design, implementation, integration, verification, validation, and configuration).
- Provides an automotive-specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs).
- Uses ASILs for specifying the item's necessary safety requirements for achieving an acceptable residual risk.
- Provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety is being achieved.^[1]

Parts of ISO 26262 [[edit](#)]

The standard consists of 9 normative parts and a guideline for the ISO 26262 as the 10th part.

GETTING READY


```
erik — -zsh
➔ ~ rustup update
info: syncing channel updates for 'stable-x86_64-apple-darwin'
info: latest update on 2019-08-15, rust version 1.37.0 (eae3437df 2019-08-13)
info: downloading component 'rustc'
 78.7 MiB / 78.7 MiB (100 %) 11.6 MiB/s ETA:  0 s
info: downloading component 'rust-std'
 56.3 MiB / 56.3 MiB (100 %) 11.0 MiB/s ETA:  0 s
info: downloading component 'cargo'
info: downloading component 'rust-docs'
info: downloading component 'rls-preview'
info: downloading component 'rust-src'
info: downloading component 'rust-analysis'
info: removing component 'rustc'
info: removing component 'rust-std'
info: removing component 'cargo'
info: removing component 'rust-docs'
info: removing component 'rls-preview'
info: removing component 'rust-src'
info: removing component 'rust-analysis'
info: installing component 'rustc'
info: installing component 'rust-std'
info: installing component 'cargo'
info: installing component 'rust-docs'
info: installing component 'rls-preview'
info: installing component 'rust-src'
```

```
rfr — -zsh
[→] rfr cargo new hello-rust
Created binary (application) 'hello-rust' package
[→] rfr ls -R | cat
hello-rust

./hello-rust:
Cargo.toml
src

./hello-rust/src:
main.rs
```




Crate chrono

See all chrono's items

Re-exports

Modules

Structs

Enums

Constants

Traits

Crates

chrono

Click or press 'S' to search, 'F' for more options...



Crate chrono

[-] [src]

[-] Chrono: Date and Time for Rust

It aims to be a feature-complete superset of the [time](#) library. In particular,

- Chrono strictly adheres to ISO 8601.
- Chrono is timezone-aware by default, with separate timezone-naive types.
- Chrono is space-optimal and (while not being the primary goal) reasonably efficient.

There were several previous attempts to bring a good date and time library to Rust, which Chrono builds upon and should acknowledge:

- [Initial research on the wiki](#)
- [Dietrich Epp's `datetime-rs`](#)
- [Luis de Bethencourt's `rust-datetime`](#)

Any significant changes to Chrono are documented in the [CHANGELOG.md](#) file.

Usage

Put this in your `Cargo.toml`:

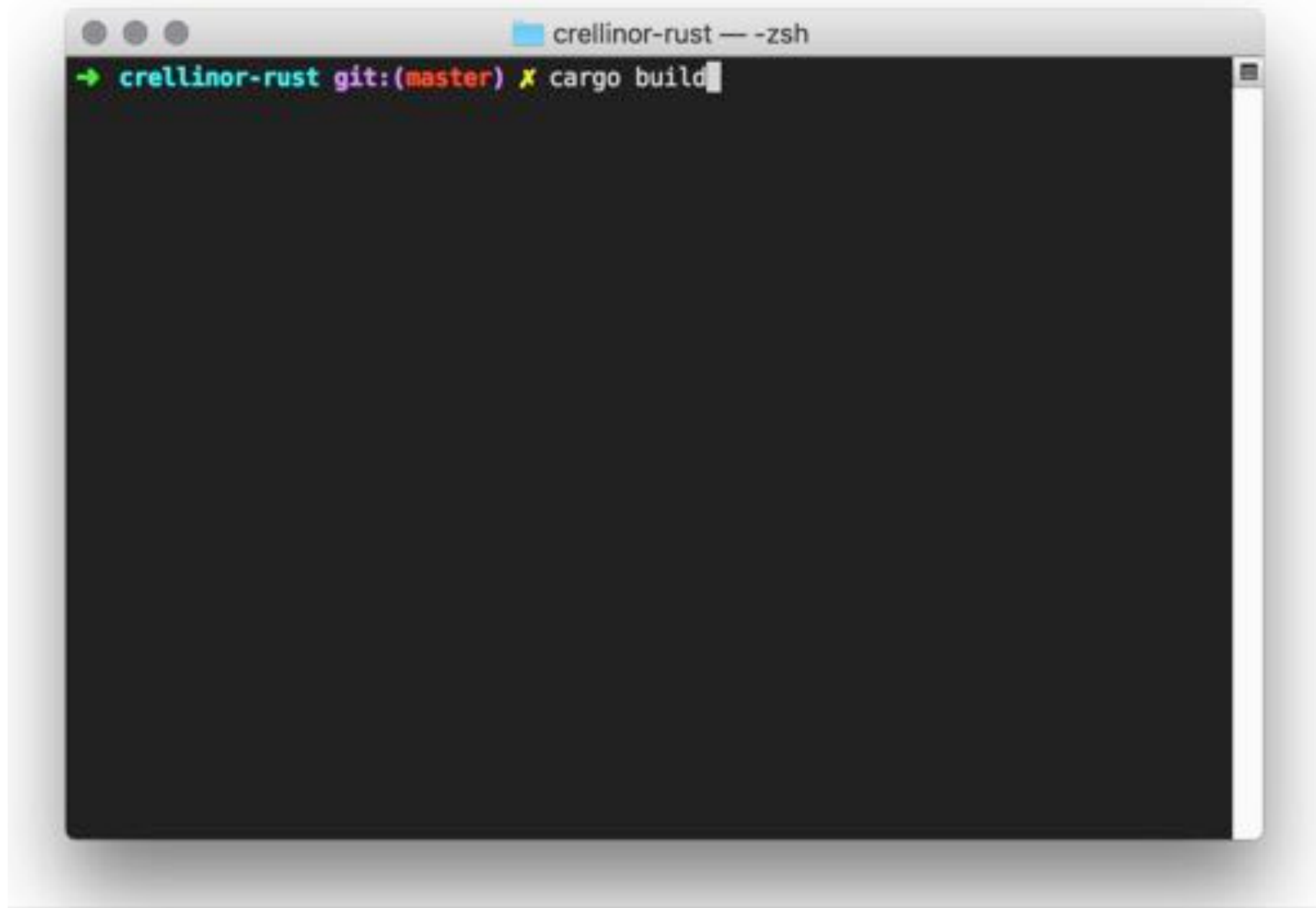
```
[dependencies]
chrono = "0.4"
```

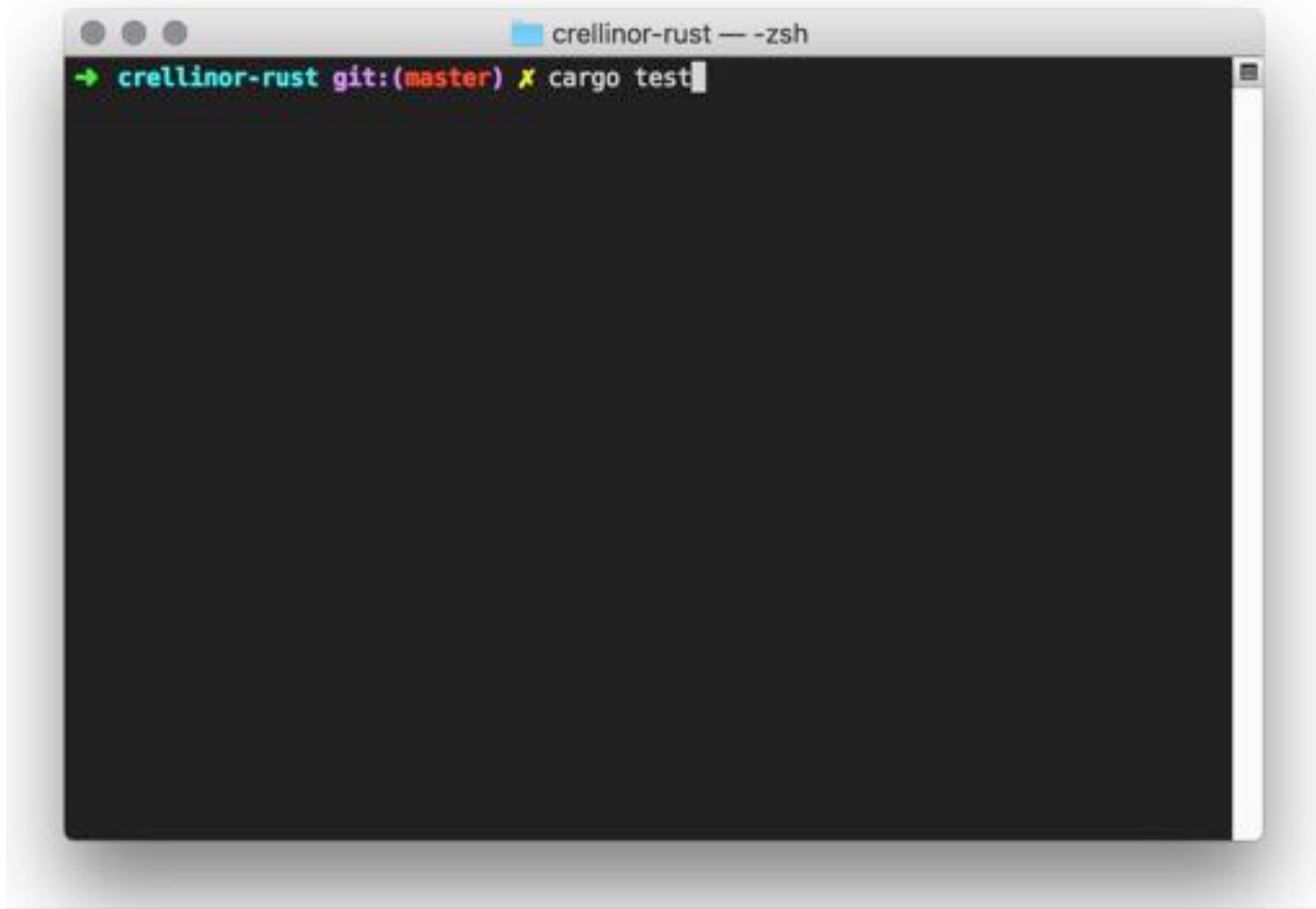
Or, if you want [serde](#) include the feature like this:

```
[dependencies]
chrono = { version = "0.4", features = ["serde"] }
```

Then put this in your crate root:

```
extern crate chrono;
```





CODE

```
15 | pub struct World {
16 |     pub name: Option<String>,
17 |     pub params: Params,
18 |     pub random: RNG,
19 |     pub terrain: Terrain,
20 |     pub cycle: u64,
21 |     pub log: Log,
22 | }
23 |
24 | impl World {
25 |     pub fn new(name: &str, params: Params) -> World {
26 |         let terrain = Terrain::with_size(params.world_size);
27 |         World {
28 |             name: Some(name.to_owned()),
29 |             params,
30 |             random: RNG::new(),
31 |             terrain,
32 |             cycle: 0,
33 |             log: Log::new(),
34 |         }
35 |     }
36 | }
```

```
156  
157     pub fn do_cycles(&mut self, num: u64) {  
158         for _ in 0..num {  
159             self.do_one_cycle();  
160         }  
161     }  
162
```

```
387  
388 pub fn cycle_count(params: &Params, prog: &[Instr]) -> u64 {  
389     prog.iter().fold(0, |acc, instr| acc + params.instr_cycles(instr))  
390 }  
391
```

```
387  
388 pub fn cycle_count(params: &Params, prog: &[Instr]) -> u64 {  
389     prog.iter().map(|instr| params.instr_cycles(instr)).sum()  
390 }  
391
```


MEMORY MANAGEMENT

```
{ // s is not valid here, it's not yet declared
  let s = "hello"; // s is valid from this point forward

  // do stuff with s
} // this scope is now over, and s is no longer valid
```

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}", world!, s1);
```



```
let s1 = String::from("hello");
```

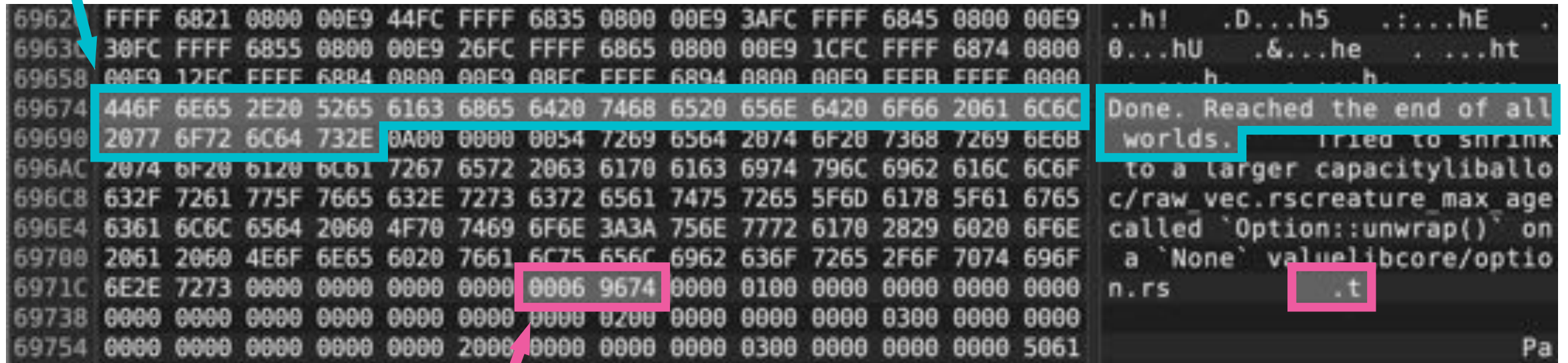
```
let len = calculate_length(&s1);
```

```
fn calculate_length(s: &String) -> usize { // s is a reference to a String  
    s.len()  
}
```

```
// Here, s goes out of scope. But because it does not have ownership of what  
// it refers to, nothing happens.
```

Illustration only! Not an accurate description of Rust, or C, or Intel CPUs.

```
let message = get_message();
```



```
let messageRef = &message;
```

```
let s1 = String::from("hello");
```

```
let len = calculate_length(&s1);
```

```
fn calculate_length(s: &String) -> usize { // s is a reference to a String
    s.len()
} // Here, s goes out of scope. But because it does not have ownership of what
// it refers to, nothing happens.
```

```
fn main() {  
    let s = String::from("hello");  
  
    change(&s);  
}  
  
fn change(some_string: &String) {  
    some_string.push_str(", world");  
}
```



```
fn main() {  
    let mut s = String::from("hello");  
  
    change(&mut s);  
}  
  
fn change(some_string: &mut String) {  
    some_string.push_str(", world");  
}
```



```
let mut s = String::from("hello");
```

```
let r1 = &mut s;
```

```
let r2 = &mut s;
```



```
let mut s = String::from("hello");
```

```
let r1 = &s; // no problem
```

```
let r2 = &s; // no problem
```

```
let r3 = &mut s; // BIG PROBLEM
```



```
fn main() {  
    let reference_to_nothing = dangle();  
}
```

```
fn dangle() -> &String {  
    let s = String::from("hello");
```

```
    &s
```

```
}
```



```
135
136 fn process_all_creatures(&mut self) {
137     self.terrain.do_with_creatures_mut(|terrain, creature, pos|
138         {
139         creature.ep -= 1;
140         if (creature.age() >= self.params.creature_max_age) || (creature.ep == 0) {
141             return None;
142         }
143         let mut ctx = PContext::new(&self.params, terrain, self.cycle, pos);
144         return Some(creature.do_cycle(&mut ctx));
145     });
146 }
147
```



```
/Users/erik/.cargo/bin/cargo build --color=always --all --all-targets
```

```
Compiling crellinor v0.1.0 (/Users/erik/Projects/GA/crellinor-rust)
```

```
error[E0501]: cannot borrow `self.terrain` as mutable because previous closure requires unique access
```

```
--> src/world.rs:137:9
```

```
137 |         self.terrain.do_with_creatures_mut(|terrain, creature, pos|
    |         ^----- closure construction occurs here
    |         |
    |         first borrow later used by call
138 |     {
139 |         creature.ep -= 1;
140 |         if (creature.age() >= self.params.creature_max_age) || (creature.ep == 0) {
    |         ---- first borrow occurs due to use of `self` in closure
...
144 |         return Some(creature.do_cycle(&mut ctx));
145 |     });
    |     ^ second borrow occurs here
```

```
error[E0500]: closure requires unique access to `self` but it is already borrowed
```

```
--> src/world.rs:137:44
```

```
137 |         self.terrain.do_with_creatures_mut(|terrain, creature, pos|
    |         ^----- closure construction occurs here
    |         |
    |         first borrow later used by call
    |         borrow occurs here
...
140 |         if (creature.age() >= self.params.creature_max_age) || (creature.ep == 0) {
    |         ---- second borrow occurs due to use of `self` in closure
```

```
error: aborting due to 2 previous errors
```

```
Some errors have detailed explanations: E0500, E0501.
For more information about an error, try `rustc --explain E0500`.
```

```
135
136 fn process_all_creatures(&mut self) {
137     self.terrain.do_with_creatures_mut(|terrain, creature, pos|
138     {
139         creature.ep -= 1;
140         if (creature.age() >= self.params.creature_max_age) || (creature.ep == 0) {
141             return None;
142         }
143         let mut ctx = PContext::new(&self.params, terrain, self.cycle pos);
144     }
```

```
135
136 fn process_all_creatures(&mut self) {
137     let cycle = self.cycle;
138     let params = &self.params;
139     self.terrain.do_with_creatures_mut(|terrain, creature, pos|
140     {
141         creature.ep -= 1;
142         if (creature.age() >= params.creature_max_age) || (creature.ep == 0) {
143             return None;
144         }
145         let mut ctx = PContext::new(params, terrain, cycle pos);
146     }
```

PARALLELISM

```
21
22 fn run_multiverse(worldfn: fn() -> World) {
23     let mut handles = Vec::new();
24
25     for tnum in 0..NUM_THREADS {
26         let h = thread::spawn(move || {
27             for snum in 0..(NUM_SIMS / NUM_THREADS) {
28                 run_world(tnum, snum, worldfn());
29             }
30         });
31         handles.push(h);
32     }
33
34     while let Some(h) = handles.pop() {
35         h.join().unwrap();
36     }
37 }
```

PERFORMANCE

The Digital Lands of Crellinor

Map | Home | Help | 100% | 100%



Clojure:
110,000 cycles/s

Rust:
~~3,500,000 cycles/s~~
25,000,000 cycles/s



“I had experienced some frustrations trying to implement in Rust the same structure I had had in C. So I mentally gave up on performance, resolving to just get something working first.”

– *Bryan Cantrill, on his blog (September 2018)*

Time to generate a statemap for a modest trace
(229M, ~3.9M state transitions)



Node.js	83.1s
---------	-------

Node/C hybrid	11.8s
---------------	-------

Rust	8.1s
------	------

ThoughtWorks®

ERIK DÖRNENBURG

HEAD OF TECHNOLOGY

erik@thoughtworks.com | @erikdoe