ferrous systems

Ferrocene is Rust for safety-critical and embedded systems.

Ferrocene is ready-to-use Rust for automotive, medical and industrial device development certified by the team that is leading in all things Rust.

Rust is safer than C/C++. Rust is built for speed with type- and memory-safe programming to prevent errors and designed to handle parallel programming. Certification now lets automotive and industrial clients take advantage of that.

Rust is critical-systems ready: TÜV SÜD conducted the audit and Ferrous Systems – leading Rust experts – supports the signed installers as well as the nightly builds. The result is Ferrocene – a quality-managed Rust compiler – certified at the highest safety standard levels.

Ferrocene is downstream from the Rust project: It works with existing Rust infrastructure and the only changes made in the code were to cover testing requirements of ISO 26262, IEC 61508 and IEC 62304 and qualification. All fixes are reported upstream for constant improvement.

Ferrocene is open source: The source code is licensed under a MIT OR Apache-2.0 license. This includes all the qualification documents. A paid license includes long-term support, validated pre-built binaries and the necessary certification for automotive and industrial use.

www.ferrocene.dev

Ferrocene is qualified by Ferrous Systems, who also:

Maintain open-source projects:

- rust-analyzer
- rust itself
- bindgen
- sudo-rs

Provide Rust training:

- Intro to Rust
- Advanced Rust
- Embedded Rust
- Management training

Do custom development:

Custom development of libraries, tools and board support packages.

Why use Rust in safety-critical systems?

```
//! Fuel monitor library.
//!
//! Provides the [`Monitor`] type to track fuel levels.
//!
//! Fuel levels are reported using the [`Reading`] type.

#![no_std]

/// Represents a single fuel reading
#[derive(Debug, Clone, Copy, PartialEq, Eq, PartialOrd, Ord)]
#[repr(C)]
pub struct Reading(pub u16);

/// Tracks fuel levels.
#[derive(Debug, Clone, Default)]
pub struct Monitor {
    /// A fixed-size buffer from the `heapless` project
    data: heapless::HistoryBuffer<Reading, 64>,
}

impl Monitor {
    /// Create a new fuel monitor, containing no readings.
    ///
    /// ```
    /// let fm = fuel::Monitor::new();
    /// ```
    pub fn new() -> Monitor {
        Default::default()
    }

    /// Add a fuel reading.
    ///
    /// ```
    /// # use fuel::{Monitor, Reading};
    /// let mut fm = Monitor::new();
    /// fm.add_reading(Reading(100));
    /// ```
    pub fn add_reading(&mut self, reading: Reading) {
        self.data.write(reading);
    }

    /// Get the mean fuel reading in the history buffer.
    ///
    /// ```
    /// use fuel::{Monitor, Reading};
    /// let mut fm = Monitor::new();
    /// assert_eq!(fm.mean_reading(), None); // No readings
    /// fm.add_reading(Reading(10));
    /// fm.add_reading(Reading(20));
    /// assert_eq!(fm.mean_reading(), Some(Reading(15)));
    /// ```
    pub fn mean_reading(&self) -> Option<Reading> {
        let sum: u64 = self.data.iter().map(|r| u64::from(r.0)).sum();
```

- Rust macros derive the tedious boilerplate code for you – without mistakes.
- Rust has rich data formatting, without heap allocation.
- Rust can use and export C functions.
- Rust's code blocks are automatically executed as part of the unit test suite. Your examples can never be out of date.

Ferrocene enables the use of Rust in safety-critical spaces. It's qualified by TÜV SÜD for use with safety-critical and functional safety systems. It is open-source and we offer long-term support on a commercial basis.

Pricing starts at €2,400.- per year for 10 seats.