ferrous systems

Ferrocene SDK for Rust:

The Rust toolkit for your safety-critical systems

By combining Rust's memory safety, performance, and concurrency features with a qualified compiler, the Ferrocene SDK for Rust is the ideal choice for professional teams working on safety-critical and embedded systems.

Ferrocene SDK for Rust is a comprehensive, professional-grade Rust distribution tailored for developers working on safety-critical, cybersecurity-sensitive, and embedded systems. It extends Rust's capabilities with tooling, long-term support, nightly builds and optional targets, providing everything you need to work with Rust effectively.

The Ferrocene SDK comes with developer tools like Knurling, a powerful toolset that provides efficient embedded logging, stack overflow detection, end-to-end traceability, and a qualified subset of core. Customization and flexibility are built-in, ensuring Rust is a perfect fit for your project requirements. For high-assurance environments, qualifications are available for ISO 26262, IEC 61508, and IEC 62304, with documentation available through our **Ferrocene Certification for Rust** product.

Advantages of the Ferrocene SDK

- **Long term support and stability:** LTS options and regular updates ensure reliability throughout your development lifecycle.
- **Comprehensive testing:** Guarantees uncompromising quality and the highest safety standards.
- **Quality managed:** Ferrocene SDK runs tests in places that the upstream project cannot, like for QNX and for bare metal, giving you the assurance you need that your toolchain is working as intended.
- **Open and supported:** Pre-built, validated binaries and tools streamline workflows while open source licensing (MIT/Apache 2.0) adds transparency
- **Built for safety:** Develop your systems with confidence using a qualified, quality-managed Rust compiler designed for safety-critical applications.

Ferrous Systems is the company leading in all things Rust:

Maintains open source projects

- Rust project contributors
- rust-analyzer
- bindgen
- sudo-rs

The builders of Ferrocene

The qualified Rust toolchain (ISO 26262, IEC 61508, IEC 62304) targets Linux, QNX and your choice of RTOS upon request. We also support bare-metal on Armv8-A and Armv7E-M.

Development support for Rust adoption

Custom development of libraries, tools and board support packages to help your teams adopt Rust.

Why use Rust in safety-critical systems?

```rust
//! 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)).
        let mean = sum.checked_div(self.data.len() as u64)?;
        Some(Reading(mean as u16))
    }
}

impl core::fmt::Display for Monitor {
    fn fmt(&self, f: &mut core::fmt::Formatter<'_>) -> core::fmt
        if let Some(mean) = self.mean_reading() {
            write!(f, "Fuel Monitor({} over {})", mean.0, self.c
```

- 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.
- Code examples in documentation blocks are automatically tested as part of Rust's test suite. Keeping your examples up to date.

Start developing safer, more efficient and resilient software with the most advanced Rust distribution available. Visit https://ferrocene.dev/ to learn more about the Ferrocene SDK and how it can transform your development process.

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