In the evolving landscape of software development, logging remains a critical co
mponent for monitoring, debugging, and ensuring the robustness of applications.
Rust, known for its safety and performance, offers various logging libraries tai
lored to its ecosystem. Among these, RustLogs (RLG) stands out as a flexible and
efficient logging framework.

Dive into RustLogs (RLG) for efficient logging in your Rust projects. This guide
covers installation, configuration, and best practices to get you started with R
LG, enhancing your Rust applications with powerful logging capabilities.

Installation

To start using RustLogs (RLG) in your Rust project, add the following line to yo
ur
file:

[dependencies]
rlg = "0.0.3"RustLogs (RLG) requires Rust 1.67.0 or later.

divider

Basic Configuration

By default, RustLogs (RLG) logs to a file named
in the current directory. You can customize the log file path by setting the
environment variable:

std::env::set_var("LOG_FILE_PATH", "/path/to/log/file.log");RustLogs (RLG) provi
des a
struct that allows you to load the configuration from environment variables or u
se default values:

use rlg::config::Config;
let config = Config::load();divider

Creating Log Entries

To create a new log entry, you can use the

function:

```
use rlg::log::Log;
use rlg::log_format::LogFormat;
use rlg::log_level::LogLevel;
let log_entry = Log::new(
"12345",
"2023-01-01T12:00:00Z",
&LogLevel::INFO,
"MyComponent",
"This is a sample log message",
&LogFormat::JSON,
);
```

The

function takes the following parameters:

A unique identifier for the session.The timestamp in ISO 8601 format.The logging level.The component generating the log.The log message.The format for the log message.divider

Logging Messages

To log a message, you can use the

method on a log entry:

```
tokio::runtime::Runtime::new().unwrap().block_on(async {
log_entry.log().await.unwrap();
});
```

RustLogs (RLG) supports asynchronous logging, allowing you to log messages without blocking your application's execution. The

method returns a

that indicates the outcome of the logging operation.

divider

Using Macros

RustLogs (RLG) provides a set of helpful macros to simplify common logging tasks . Here are a few examples:

Creates a new log entry with specified parameters.let log = macro_log!(session_id, time, level, component, description, format);Creates an info log with default session ID and format.let log = macro_info_log!(time, component, description);Prints a log to stdout.macro_print_log!(log);Asynchronously logs a message to a file.let result = macro_log_to_file!(log);For a complete list of available macros and their usage, refer to the RustLogs (RLG) Documentation .

divider

Error Handling

Errors can occur during logging operations, such as file I/O errors or formatting errors. The

method returns a

that indicates the outcome of the logging operation. You should handle potential errors appropriately in your code:

match log_entry.log().await {

Ok(_) => println!("Log entry successfully written"),

Err(err) => eprintln!("Error logging entry{}", err),

}divider

Next Steps

Congratulations! You're now ready to start using RustLogs (RLG) in your Rust pro

jects. Here are some next steps you can take:

Customize the log format and output destinations to suit your application's need s.Integrate RustLogs (RLG) into your existing Rust projects and start logging me ssages effectively.Contribute to the RustLogs (RLG) Repository by reporting iss ues, suggesting improvements, or submitting pull requests.If you have any questi ons or need further assistance, check our FAQs and feel free to reach out to th e RustLogs (RLG) community or open an issue on the GitHub repository .

Happy logging with RustLogs (RLG)!

divider