TRN-Rust: High-Performance Tool Resource Name Library

rust 1.70+

license MIT

docs not found

A high-performance Rust library for parsing, validating, and manipulating Tool Resource Names (TRN) in AI Agent platforms. This library provides enterprise-grade functionality with memory safety, type safety, and exceptional performance.

🚀 Features

- **High Performance**: 100K+ TRN parses per second
- **Memory Safe**: Zero-copy parsing with compile-time guarantees
- **Type Safe**: Strong typing with comprehensive validation
- **Thread Safe**: Concurrent operations with shared state
- **Enterprise Ready**: Comprehensive error handling and monitoring
- **Multiple Formats**: Support for TRN strings, URLs, JSON, YAML
- **Pattern Matching**: Advanced filtering and search capabilities
- **Builder Pattern**: Fluent API for TRN construction
- **CLI Tools**: Command-line interface for TRN operations
- **Extensive Testing**: 100+ unit and integration tests

📋 TRN Format

```
trn:platform[:scope]:resource_type:type[:subtype]:instance_id:version[:tag][@hash]
```

Components

Component	Description	Required	Examples
`platform`	Platform identifier	✅	`user`, `org`, `aiplatform`
`scope`	User/organization scope	🔍 *	`alice`, `company`, `team-dev`
`resource_type`	Type of resource	✅	`tool`, `model`, `dataset`, `pipeline`
`type`	Specific tool type	📝	`openapi`, `workflow`, `python`, `shell`
`subtype`	Tool subtype	❌	`async`, `streaming`, `batch`
`instance_id`	Unique identifier	✅	`github-api`, `bert-base`
`version`	Resource version	✅	`v1.0`, `latest`, `v2.1-beta`

Component	Description	Required	Examples
`tag`	Environment tag	❌	`stable`, `beta`, `production`
`hash`	Content hash	❌	`abc123def456`

*Required for `user` and `org` platforms, optional for `aiplatform`

🏃 Quick Start

Installation

Add to your `Cargo.toml`:

```toml
[dependencies]
trn-rust = "0.1.0"
```

Basic Usage

```rust
use trn_rust::{Trn, TrnBuilder, Platform, ResourceType, ToolType};

// Parse existing TRN
let trn = Trn::parse("trn:user:alice:tool:openapi:github-api:v1.0")?;
println!("Platform: {:?}", trn.platform());
println!("Instance: {}", trn.instance_id());

// Build new TRN
let trn = TrnBuilder::new()
    .platform(Platform::User)
    .scope("alice")
    .resource_type(ResourceType::Tool)
    .tool_type(ToolType::OpenApi)
    .instance_id("github-api")
    .version("v1.0")
    .build()?;

// Validate and convert
trn.validate()?;
let url = trn.to_url()?;
println!("URL: {}", url); // trn://user/alice/tool/openapi/github-
api/v1.0
```

📚 Examples

The `examples/` directory contains comprehensive usage examples:

Basic Operations

```
# Run basic usage examples
cargo run --example basic_usage
```

Advanced Pattern Matching

```
# Run advanced pattern examples
cargo run --example advanced_patterns
```

Command Line Interface

```
# Parse and validate TRNs
cargo run --example cli_usage -- parse
"trn:user:alice:tool:openapi:github-api:v1.0"
cargo run --example cli_usage -- validate
"trn:user:alice:tool:openapi:github-api:v1.0"

# Convert formats
cargo run --example cli_usage -- convert
"trn:user:alice:tool:openapi:github-api:v1.0" url
cargo run --example cli_usage -- convert
"trn:user:alice:tool:openapi:github-api:v1.0" json

# Interactive builder
cargo run --example cli_usage -- build

# Batch processing
cargo run --example cli_usage -- batch sample_trns.txt
```

Performance Testing

```
# Run performance benchmarks (use release mode)
cargo run --example performance_testing --release
```

🎯 Core API

Parsing and Validation

```
// Parse TRN string
let trn = Trn::parse("trn:user:alice:tool:openapi:github-api:v1.0")?;

// Validate business rules
trn.validate()?;
```

```
// Access components
println!("Platform: {:?}", trn.platform());
println!("Scope: {:?}", trn.scope());
println!("Version: {}", trn.version());
```

Builder Pattern

```
let trn = TrnBuilder::new()
    .platform(Platform::Org)
    .scope("company")
    .resource_type(ResourceType::Tool)
    .tool_type(ToolType::Workflow)
    .subtype("async")
    .instance_id("user-onboarding")
    .version("v2.1")
    .tag("production")
    .build()?;
```

URL Conversion

```
// Convert to TRN URL format
let url = trn.to_url()?;
// Result: "trn://user/alice/tool/openapi/github-api/v1.0"

// Convert to HTTPS URL
let https_url = trn.to_https_url("https://api.example.com")?;

// Parse from URL
let trn_from_url = Trn::from_url("trn://user/alice/tool/openapi/github-
api/v1.0")?;
```

Format Conversion

```
// Export to different formats
let json = trn.to_json()?;
let yaml = trn.to_yaml()?;

// Parse from JSON
let trn: Trn = serde_json::from_str(&json_string)?;
```

Pattern Matching and Filtering

```
// Find all tools by Alice
let alice_tools: Vec<_> = trns.iter()
    .filter(|trn| trn.scope() == Some("alice"))
    .collect();

// Find OpenAPI tools
let openapi_tools: Vec<_> = trns.iter()
    .filter(|trn| trn.tool_type() == Some(&ToolType::OpenApi))
    .collect();

// Complex filtering
let stable_user_tools: Vec<_> = trns.iter()
    .filter(|trn| {
        trn.platform() == &Platform::User &&
        trn.tag() == Some("stable")
    })
    .collect();
```

⚡ Performance

Performance benchmarks on modern hardware:

Operation	Performance	Notes
Parsing	100K+ TRNs/sec	Zero-copy parsing
Building	50K+ TRNs/sec	Builder pattern
Validation	200K+ validations/sec	With caching
URL Conversion	150K+ conversions/sec	Bidirectional
Concurrent Ops	High throughput	Thread-safe operations

Running Benchmarks

```
# Run official benchmarks
cargo bench

# Run performance examples
cargo run --example performance_testing --release
```

🔧 CLI Tool

The library includes a comprehensive CLI tool for TRN operations:

```
# Built-in CLI commands
cargo run --bin trn -- parse "trn:user:alice:tool:openapi:github-
```

```
api:v1.0"
cargo run --bin trn -- validate "trn:user:alice:tool:openapi:github-
api:v1.0"
cargo run --bin trn -- convert "trn:user:alice:tool:openapi:github-
api:v1.0" --format json

# Process files
echo "trn:user:alice:tool:openapi:github-api:v1.0" | cargo run --bin trn
-- validate --stdin
cargo run --bin trn -- batch --file sample_trns.txt
```

📖 Documentation

- **API Documentation**: Run `cargo doc --open` to view comprehensive API docs
- **Examples**: See `examples/` directory for detailed usage patterns
- **Architecture**: See `RUST_DESIGN.md` for design decisions and architecture
- **Performance**: See benchmarks and performance examples

🧪 Testing

```
# Run all tests
cargo test

# Run with output
cargo test -- --nocapture

# Run specific test module
cargo test test_parsing

# Run integration tests
cargo test --test integration_tests

# Run with coverage (requires cargo-tarpaulin)
cargo tarpaulin --out html
```

Test Coverage

The library includes comprehensive testing:

- **Unit Tests**: 70+ tests covering all modules
- **Integration Tests**: End-to-end functionality testing
- **Property Tests**: Fuzzing and edge case testing
- **Performance Tests**: Benchmarks and performance validation
- **Concurrent Tests**: Thread safety validation

🔍 Error Handling

The library provides detailed error information with suggestions:

```
match Trn::parse(trn_string) {
    Ok(trn) => {
        match trn.validate() {
            Ok(()) => println!("Valid TRN: {}", trn),
            Err(e) => {
                eprintln!("Validation error: {}", e);
                // Error includes suggestions for fixes
            }
        }
    }
    Err(e) => {
        eprintln!("Parse error: {}", e);
        // Detailed error with position and expected format
    }
}
```

Error Types

- `TrnError::InvalidFormat`: Malformed TRN string
- `TrnError::ValidationFailed`: Business rule violations
- `TrnError::InvalidComponent`: Invalid component values
- `TrnError::BuilderError`: Builder pattern errors
- `TrnError::ConversionError`: Format conversion errors

🌟 Advanced Features

Caching and Performance

```
use trn_rust::validation::{TrnValidator, ValidationConfig};

// Configure validation caching
let validator = TrnValidator::with_config(ValidationConfig {
    cache_ttl: Duration::from_secs(3600),
    max_cache_size: 10000,
    enable_caching: true,
});

// Reuse validator for high-performance validation
for trn in trns {
    validator.validate(&trn)?;
}
```

Custom Types

```
// Support for custom platforms and types
let trn = TrnBuilder::new()
```

```
    .platform(Platform::Custom("enterprise".to_string()))
    .resource_type(ResourceType::Custom("workflow".to_string()))
    // ... other components
    .build()?;
```

Batch Operations

```
use trn_rust::utils::{batch_parse, batch_validate};

// Batch parsing with error collection
let results = batch_parse(&trn_strings);
println!("Parsed: {}, Failed: {}", results.successes.len(),
results.failures.len());

// Batch validation with statistics
let validation_results = batch_validate(&trns);
println!("Success rate: {:.1}%",
         validation_results.success_rate() * 100.0);
```

🤝 Contributing

Contributions are welcome! Please see CONTRIBUTING.md for guidelines.

Development Setup

```
# Clone the repository
git clone <repository-url>
cd trn-rust

# Install dependencies and tools
cargo install cargo-tarpaulin  # For coverage
cargo install cargo-criterion  # For benchmarking

# Run development checks
cargo check
cargo test
cargo clippy
cargo fmt

# Run benchmarks
cargo bench
```

Project Structure

```
trn-rust/
├── src/
│   ├── lib.rs              # Main library entry point
│   ├── types.rs            # Core TRN types and structures
│   ├── parsing.rs          # TRN parsing logic
│   ├── validation.rs       # Validation and caching
│   ├── builder.rs          # Builder pattern implementation
│   ├── url.rs              # URL conversion functionality
│   ├── pattern.rs          # Pattern matching utilities
│   ├── utils.rs            # Utility functions
│   ├── constants.rs        # Constants and regex patterns
│   ├── error.rs            # Error types and handling
│   └── bin/
│       └── trn.rs          # CLI application
├── examples/               # Usage examples
├── tests/                  # Integration tests
├── benches/               # Performance benchmarks
└── docs/                  # Additional documentation
```

📄 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- Inspired by AWS ARN format for resource identification
- Built with Rust's powerful type system and memory safety
- Designed for AI Agent platform requirements
- Performance optimized for high-throughput scenarios

📞 Support

- **Issues**: GitHub Issues
- **Documentation**: docs.rs/trn-rust
- **Examples**: See `examples/` directory
- **Performance**: See benchmarks and performance guides

Built with ❤️ and Rust for AI Agent platforms