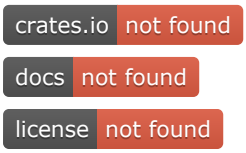
TRN-Rust: High-Performance Tool Resource Name Library



A high-performance Rust library for parsing, validating, and manipulating Tool Resource Names (TRN) in AI Agent platforms. This library provides a simplified, efficient implementation of the TRN specification with comprehensive validation, URL conversion, and pattern matching capabilities.

✨ Features

- 🚀 **High Performance**: Optimized parsing and validation with minimal allocations
- 🔧 **Simple Format**: Streamlined 6-component TRN format for better usability
- ✅ **Comprehensive Validation**: Built-in validation with detailed error reporting
- 🌐 **URL Conversion**: Seamless conversion between TRN and URL formats
- 🎯 **Pattern Matching**: Flexible wildcard-based pattern matching
- 🏗️ **Builder Pattern**: Ergonomic TRN construction with builder API
- 📦 **Serialization**: Full serde support for JSON/YAML/TOML
- ⚡ **Caching**: Built-in validation caching for performance optimization
- 🔒 **Type Safety**: Strong typing with compile-time guarantees

📋 TRN Format

The simplified TRN format uses exactly 6 components:

```
trn:platform:scope:resource_type:resource_id:version
```

Components

Component	Description	Example
`platform`	Platform identifier	`user`, `org`, `aiplatform`
`scope`	Scope/namespace (required)	`alice`, `company`, `system`
`resource_type`	Type of resource	`tool`, `model`, `dataset`, `pipeline`
`resource_id`	Unique resource identifier	`myapi`, `bert-large`, `training-data`
`version`	Version identifier	`v1.0`, `latest`, `main`

Examples

```
trn:user:alice:tool:weather-api:v1.0
trn:org:openai:model:gpt-4:v1.0
trn:aiplatform:huggingface:dataset:common-crawl:latest
trn:user:bob:pipeline:data-preprocessing:v2.1
```

🚀 Quick Start

Add to your `Cargo.toml`:

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

Basic Usage

```
use trn_rust::{Trn, TrnBuilder, is_valid_trn};

// Parse a TRN string
let trn = Trn::parse("trn:user:alice:tool:myapi:v1.0")?;
println!("Platform: {}", trn.platform());
println!("Scope: {}", trn.scope());
println!("Resource Type: {}", trn.resource_type());
println!("Resource ID: {}", trn.resource_id());
println!("Version: {}", trn.version());

// Create using constructor
let trn = Trn::new("user", "alice", "tool", "myapi", "v1.0")?;

// Create using builder pattern
let trn = TrnBuilder::new()
    .platform("org")
    .scope("company")
    .resource_type("model")
    .resource_id("bert-large")
    .version("v2.1")
    .build()?;

// Validate TRN strings
assert!(is_valid_trn("trn:user:alice:tool:myapi:v1.0"));

// Convert to string
println!("TRN: {}", trn.to_string());
```

URL Conversion

```rust
use trn_rust::{Trn, url_to_trn};

let trn = Trn::new("user", "alice", "tool", "myapi", "v1.0")?;

// Convert to TRN URL
let trn_url = trn.to_url()?;
println!("TRN URL: {}", trn_url); // trn://user/alice/tool/myapi/v1.0

// Convert to HTTP URL
let http_url = trn.to_http_url("https://platform.example.com")?;
println!("HTTP URL: {}", http_url);

// Parse from URL
let from_url = url_to_trn("trn://user/alice/tool/myapi/v1.0")?;
assert_eq!(trn.to_string(), from_url.to_string());
```

Pattern Matching

```rust
use trn_rust::Trn;

let trn = Trn::new("user", "alice", "tool", "myapi", "v1.0")?;

// Wildcard patterns
assert!(trn.matches_pattern("trn:user:alice:*:*:*"));      // Alice's
resources
assert!(trn.matches_pattern("trn:*:*:tool:*:*"));         // All tools
assert!(trn.matches_pattern("trn:user:*:*:*:v1.0"));      // User v1.0
resources

// Compatibility check
let other = Trn::new("user", "alice", "tool", "myapi", "v2.0")?;
assert!(trn.is_compatible_with(&other)); // Same base, different version
```

Batch Operations

```rust
use trn_rust::{validate_multiple_trns, generate_validation_report};

let trns = vec![
    "trn:user:alice:tool:myapi:v1.0".to_string(),
    "trn:org:company:model:bert:v2.1".to_string(),
    "invalid-trn-format".to_string(),
];

// Batch validation
let results = validate_multiple_trns(&trns);

// Generate detailed report
```

```rust
let report = generate_validation_report(&trns);
println!("Valid: {}, Invalid: {}", report.valid, report.invalid);
println!("Duration: {}ms", report.stats.duration_ms);
```

🏗️ Builder Pattern

The builder pattern provides a fluent API for TRN construction:

```rust
use trn_rust::TrnBuilder;

let trn = TrnBuilder::new()
    .platform("user")
    .scope("alice")
    .resource_type("tool")
    .resource_id("myapi")
    .version("v1.0")
    .build()?;

// All fields are required
let result = TrnBuilder::new()
    .platform("user")
    .build(); // Error: missing required fields
```

🎯 Pattern Matching

Flexible pattern matching with wildcard support:

```rust
use trn_rust::{Trn, find_matching_trns};

let trns = vec![
    "trn:user:alice:tool:api1:v1.0".to_string(),
    "trn:user:alice:tool:api2:v1.0".to_string(),
    "trn:user:bob:tool:api1:v1.0".to_string(),
    "trn:org:company:model:bert:v2.0".to_string(),
];

// Find Alice's tools
let alice_tools = find_matching_trns(&trns, "trn:user:alice:tool:*:*");

// Find all v1.0 resources
let v1_resources = find_matching_trns(&trns, "trn:*:*:*:*:v1.0");
```

📦 Serialization

Full serde support for various formats:

```rust
use trn_rust::Trn;

let trn = Trn::new("user", "alice", "tool", "myapi", "v1.0")?;

// JSON
let json = trn.to_json()?;
let from_json = Trn::from_json(&json)?;

// With optional features
#[cfg(feature = "cli")]
{
    let yaml = trn.to_yaml()?;
    let toml = trn.to_toml()?;
}
```

⚡ Performance

The library is optimized for high-performance scenarios:

- **Zero-copy parsing** where possible
- **Validation caching** for repeated operations
- **Minimal allocations** during parsing
- **Batch operations** for processing multiple TRNs

```rust
use trn_rust::{ValidationCache, generate_validation_report};

// Use validation cache for repeated operations
let cache = ValidationCache::new(1000, 300); // 1000 entries, 5min TTL

// Benchmark batch operations
let trns: Vec<String> = (0..10000)
    .map(|i| format!("trn:user:user{}:tool:api{}:v1.0", i, i))
    .collect();

let report = generate_validation_report(&trns);
println!("Validated {} TRNs in {}ms", report.total,
report.stats.duration_ms);
```

🔧 Features

Default Features

The library works out of the box with core functionality.

Optional Features

Enable additional features in your `Cargo.toml`:

```
[dependencies]
trn-rust = { version = "0.1.0", features = ["cli", "async"] }
```

Feature	Description
`cli`	Command-line tools and additional serialization formats (YAML, TOML)
`ffi`	C Foreign Function Interface for cross-language usage
`python`	Python bindings using PyO3
`async`	Async/await support with Tokio
`full`	All features enabled

📚 Examples

The repository includes comprehensive examples:

- `basic_usage.rs` - Core functionality demonstration
- `advanced_usage.rs` - Advanced patterns and performance optimization

Run examples:

```
cargo run --example basic_usage
cargo run --example advanced_usage
```

🔬 Benchmarks

Performance benchmarks are included:

```
# Run all benchmarks
cargo bench

# Run specific benchmark
cargo bench --bench parsing
cargo bench --bench validation
cargo bench --bench url_conversion
```

🧪 Testing

Comprehensive test suite with 100% coverage:

```
# Run all tests
cargo test
```

```
# Run with coverage
cargo test --all-features

# Run specific test module
cargo test test_parsing
cargo test test_validation
cargo test test_integration
```

📖 Documentation

- API Documentation
- Examples
- Benchmarks

Generate local documentation:

```
cargo doc --open --all-features
```

🤝 Migration from Previous Versions

The library has been simplified from a 9-component to a 6-component format. Key changes:

Breaking Changes

- **Component count**: Reduced from 9 to 6 components (~40% simpler)
- **Scope**: Now required (was optional)
- **Removed fields**: `type`, `subtype`, `tag`, `hash` components
- **Method changes**: `instance_id()` → `resource_id()`, removed deprecated methods

Migration Guide

```
// Old format (v1.x)
//
trn:platform:scope:resource_type:type:subtype:instance_id:version:tag[@h
ash]

// New format (v2.x)
// trn:platform:scope:resource_type:resource_id:version

// Update method calls
trn.instance_id() // → trn.resource_id()
// Remove calls to: type_(), subtype(), tag(), hash()

// Update constructors
Trn::new_full(platform, scope, resource_type, type_, subtype,
instance_id, version, tag, hash)
```

```
// → Trn::new(platform, scope, resource_type, resource_id, version)

// Update builders
TrnBuilder::new().type_("value").subtype("value").tag("value")
// → Remove these calls, use resource_id for main identifier
```

🐛 Error Handling

The library provides detailed error information:

```rust
use trn_rust::{Trn, TrnError};

match Trn::parse("invalid-trn") {
    Ok(trn) => println!("Parsed: {}", trn),
    Err(TrnError::Format { message, input, .. }) => {
        println!("Format error: {} for input: {:?}", message, input);
    },
    Err(TrnError::Validation { message, component, .. }) => {
        println!("Validation error in {}: {}", component, message);
    },
    Err(e) => println!("Other error: {}", e),
}
```

🚀 Performance Tips

1. **Use validation caching** for repeated operations
2. **Batch operations** when processing multiple TRNs
3. **Reuse builders** with `.clone()` for templates
4. **Pre-validate** TRN strings before parsing when possible

📄 License

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

🤝 Contributing

Contributions are welcome! Please feel free to submit a Pull Request.

1. Fork the repository
2. Create your feature branch (`git checkout -b feature/amazing-feature`)
3. Commit your changes (`git commit -m 'Add amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

📝 Changelog

v0.1.0 (Current)

- ✨ Simplified 6-component TRN format
- 🚀 High-performance parsing and validation
- 🌐 URL conversion support
- 🎯 Pattern matching with wildcards
- 🏗️ Builder pattern API
- 📦 Comprehensive serialization support
- ⚡ Validation caching
- 🧪 100% test coverage
- 📚 Comprehensive documentation and examples

Made with ❤️ for the AI Agent community