DateTime (DTT) Setup and Usage Guide

Overview

is a versatile Rust library designed for parsing, validating, manipulating, and formatting dates and times. Tailored to modern software development needs, enhances date and time data handling with a focus on performance and accuracy. It offers a range of functions and data structures that allow you to perform var ious date and time operations with ease, such as determining the day of the mont h, hour of the day, working with ISO 8601 date and time formats, and many others .

The library supports the creation of new DateTime objects with either UTC or cus tom timezone specifications, ensuring that you always have accurate and relevant date and time information. Additionally, it provides a mechanism to validate inp ut dates and times, ensuring that you always have accurate information to work w ith.

Features

Core Features

Versatile Parsing and FormattingEfficiently handles various date and time format s.Rigorous ValidationEnsures data integrity and correctness.Advanced Manipulatio nFacilitates complex date and time computations and transformations.Rust-Optimiz edLeverages Rust's capabilities for high-performance and safety. Advanced Featur es

ReliabilityImplements error handling for robust and secure operations.Configurab ilityAdaptable to different programming requirements.Comprehensive Documentation Well-documented API for easy integration and use. Functionality

Parsing and ValidationParse and validate date and time data efficiently.Manipula

tion ToolsPerform a range of date and time manipulations.Flexible FormattingCustomize the representation of date and time data.Error HandlingRobust handling of common date and time processing errors.The library
provides date and time types and methods to make it easier to manipulate dates and times. It uses the serde library to derive the Deserialize and Serialize traits to convert the
struct to and from various data formats. It also uses the time and regex crates to deal with time conversions and regular expressions respectively.
The
struct includes fields such as:

Feature	Description
	Day of the month(01-31)
	Hour of the day(00-23)
	ISO 8601 date and time(e.g. "2023-01-01T00:00:00+00:00")
	ISO week number(1-53)
	Microsecond(0-999999)
	Minute of the hour(0-59)

| Month(e.g. "January") |

|

| Now object(e.g. "2023-01-01") |

|

| Offset from UTC(e.g. "+00:00") |

|

| Ordinal date(1-366) |

|

| Second of the minute(0-59) |

|

| Time object(e.g. "00:00:00") |

|

| Time zone object(e.g. "UTC") |

|

| Weekday object(e.g. "Monday") |

|

| Year object(e.g. "2023") |

of which represents different aspects of a date and time.

The

struct has two methods to create instances:

and

.

creates a new

object with UTC timezone, and

creates a new

object with a custom timezone.

It also includes a method

which checks if the input string represents a valid day of the week. It also inc

ludes a method

which checks if the input string represents a valid month of the year.

Started

Begin using

in just a few steps.

Requirements

Ensure you have the Rust toolchain,version 1.69.0 or later (stable).

Installation

After installing the Rust toolchain (instructions on the Rust website), instal

l DateTime (DTT) with:

instructions on the Rust website .

Once you have the Rust toolchain installed, you can install

using the following command:

cargo install dttYou can then run the help command to see the available options:

dtt --help

To use the

library in your project, add the following to your

file:

[dependencies]

dtt = "0.0.4"Add the following to your

file:

extern crate dtt;

use dtt::*;then you can use the functions in your application code.

defines specific error types, such as

and

, ensuring robust error handling in your applications.

To get started with

, you can use the examples provided in the

directory of the project.

To run the examples, clone the repository and run the following command in your terminal from the project root directory.

cargo run --example dtt Example 1Creating a new DateTime object

```
use dtt::DateTime;
use dtt::dtt_print;
fn main() {
// Create a new DateTime object with the current UTC time
let now = DateTime::new();
dtt_print!(now);
```

} Example 2Creating a new DateTime object with a custom timezone

```
use dtt::DateTime;
use dtt::dtt_print;
fn main() {
// Create a new DateTime object with a custom timezone (e.g., CEST)
let paris_time = DateTime::new_with_tz("CEST");
dtt_print!(paris_time);
```

}Custom timezone supported by

are:

Abbreviation	UtcOffset	Time Zone Description
ACDT		Australian Central Daylight Time
ACST		Australian Central Standard Time
ADT		Atlantic Daylight Time
AEDT		Australian Eastern Daylight Time
AEST		Australian Eastern Standard Time
AKDT		Alaska Daylight Time
AKST		Alaska Standard Time
AST		Atlantic Standard Time
AWST		Australian Western Standard Time
BST		British Summer Time
CDT		

| Central Daylight Time |

| CEST |

| Central European Summer Time |

| CET |

| Central European Time |

| CST |

| Central Standard Time |

| ECT |

| Eastern Caribbean Time |

| EDT |

| Eastern Daylight Time |

| EEST |

| Eastern European Summer Time |

| EET |

| Eastern European Time |

| EST |

| Eastern Standard Time |

| GMT |

| Greenwich Mean Time |

| HADT |

| Hawaii-Aleutian Daylight Time |

| HAST |

| Hawaii-Aleutian Standard Time |

| HKT |

| Hong Kong Time |

| IST |

| Indian Standard Time |

| IDT |

| Israel Daylight Time |

| JST |

| Japan Standard Time |

| KST |

| Korean Standard Time |

| MDT |

| Mountain Daylight Time |

| MST |

| Mountain Standard Time |

| NZDT |

| New Zealand Daylight Time |

| NZST |

| New Zealand Standard Time |

| PDT |

| Pacific Daylight Time |

| PST |

| Pacific Standard Time |

| UTC |

| Coordinated Universal Time |

| WADT |

| West Australian Daylight Time |

| WAST |

| West Australian Standard Time |

| WEDT |

| Western European Daylight Time |

| WEST |

| Western European Summer Time |

| WET |

| Western European Time |

| WST |

| Western Standard Time |

3Formatting a DateTime object

```
use dtt::DateTime;
use dtt::dtt_print;
fn main() {
// Create a new DateTime object with the current UTC time
let now = DateTime::new();
// Format the DateTime object as a string
let formatted_time = now.format("%Y-%m-%d %H:%M:%S");
dtt_print!("Formatted time{}", formatted_time);
```

} Example 4Parsing a string into a DateTime object

```
use dtt::DateTime;
use dtt::dtt_print;
fn main() {
// Parse a string into a DateTime object
let date_string = "2023-05-12T12:00:00+00:00";
match DateTime::parse(date_string) {
```

```
Ok(datetime) => dtt_print!("Parsed DateTime{}", datetime),
Err(err) => dtt_print!("Error parsing DateTime{}", err),
}
}
```

For comprehensive information and resources related to
, we invite you
to explore our extensive documentation.
Detailed guides, API references, and examples are available on docs.rs ,
where you'll find in-depth material to support your development needs.
For library-specific details and to access our collection of Rust libraries,
visit lib.rs .
Additionally, you can explore crates.io to find a wide range of Rust
crates, which include the necessary tools and libraries to enhance your
projects.
These resources are designed to provide you with the most up-to-date
and thorough information to assist in your development endeavours.
!Divider