Rust and Unicode

Behnam Esfahbod

Lightning Talk — Internationalization & Unicode Conference 41 — October 17, 2017

Firefox is using `encoding_rs` since 56.0 release, it means you are now using Rust in Firefox.
Rust is a systems programming language that has many benifits, one is how Unicode-friend it is.
Natively, Rust supports codepoint-based numeric `char` type that represents Unicode Scalar Values.
Strings are represented natively in UTF-8 (`str`), which can be viewed as a sequence of bytes or chars, as needed.
There are many Unicode-related *cargo* packages already available on *crates.io*, the Rust package host.
UNIC (Unicode and Internationalization Crates for Rust) has the ambitious goal to be ICU for Rust.
Many UCD character properties are already made available in UNIC, as well as
Unicode Bidirectional Algorithm, Normalization Forms, and IDNA solutions.
UNIC is expanding with new components and looking for new contributors to join the project.

https://github.com/behnam/rust-unic

��� ‖�

����� ����� ������� � ��������� ����� �� ������

����� ��� ���� �� ��� ���� ���� ���� ���� ����� ���� �� ����� ����� ������� � ��������� �����
.������� (����)��� ������������� ��� � ����� �����
��������� � ����� ��������� �� ���� ��� ����������� ���� ����� �������� �� ��������� �����
��� ��� �� ��� .������� ���� ����� �� ����� � �� ������� � ������ ������� ����� ������ � �����
���� �� � ��� ������� ����� �� ���� �� � ��� ����� � ���������� ����� ���� ����������� ��
:���� ��� � �� ������ ������� � ����� ������ �� ����
β�� ���� ���β�� ���è���� ����� ���� � ����� � ������ � ��������� ����� ����. ������ �����
���� ������������ ��� ��� �������� ���� ����� ����� � ����� ���������� ��� ����� �����
������. � �� �� ������ ����� � ��������� �� ����� ��� ����� �� �� ���� ��� � ������ ����� ��
.������ ����� � ����� ��������� ���(����) �‖
����� ����� ����� �� � ������ � ������� ���������� � ������ ���� �� ������������� �������
ḓ �� ����� ��������� ������� ������ � ����� ��� ����� �� ����� � ��� �� �� ��� ������ ���� ���
�� ����� ���� ���� �� �� ������ ���� ��� ������������ ����� ���� .������� ����� � �‖
����� ����� �� ������� ����� ����� ����� �� ����� ��� ����� �� � ����� �5� ����� ���� ���������
.����
�� ������� ��������� ��������� ������ ���� ����� �� ����� ����� ������� � ��������� �����
���������� ����� �� w��� ��� ������������ � ��� ����� �� ���� ��� ��� ������������ ����� ��
.�������� ������� ���������� ������������ � ���� ����� � ��� �����
�� ����� �� ��� �������� ������ ����� �� ��������� ����� ����� ������� � ��������� �����
β�� ��� ����� � ����� ���� � ������ ���� ������� �� ����� ������� ���� ���� �� ����� �����
������� ���� ����� � �������� ����� ���� �� ������������ �� �� ����� ����� ���� ��������
����� ������ �����. ����� �������� �� ����� ��� �������� ��������� ���� ������� ���w
.����� ������ �� �� ���� ���� � ������� ������� ���������
β�� �� ����� ��������� � ������ ���� �� �������� ������� �� ������ ������� ����� �� �����
������ ����ɲ ������� ����� � ����� ���β�� ��������� ������ ����� � ������ ���� ������
�����ɛ����� � ������� ���β�� ����� ���ɲ ����� ��������� ������ ��������� � ������ �� �� ����
����� ���� ����� ������ ����� ���� ������� ���� ���� � �� ���� ����� ����� ���� ����� ����
����� ������ �� ���� ���� ����� � �� ������ ���� ����� ��������. ����� ����� ���������� ���ā
��� ������ ���� �������� ������� ����� � ������� ����� ���� ����� ����������� ����� ���ɲ
.������� ����� ��� ����� ��� �� ��� ������������

جمهوری اسلامی ایران

موسسه استاندارد و تحقیقات صنعتی ایران

شمارهٔ استاندارد ایران

6219

[illegible]

[illegible]

[illegible]

06F7	رقمِ فارسیِ هفت	202A*	زیرمتنِ چپ‌به‌راست
06F8	رقمِ فارسیِ هشت	202B*	زیرمتنِ راست‌به‌چپ
06F9	رقمِ فارسیِ نه	202C*	پایانِ زیرمتن
200C	فاصله‌ی مجازی	202D*	زیرمتنِ اکیداً چپ‌به‌راست
200D	اتصالِ مجازی	202E*	زیرمتنِ اکیداً راست‌به‌چپ
200E	نشانه‌ی چپ‌به‌راست	2212*	علامتِ منها
200F	نشانه‌ی راست‌به‌چپ	FEFF	نشانه‌ی ترتیبِ بایت‌ها

.���� ����� �� ���� ������� ����� �� ����� �� ���� ��� �� ����� ����� - �������

ISLAMIC REPUBLIC OF IRAN

Institute of Standards and Industrial Research of Iran

ISIRI NUMBER

6219

Information technology - Persian information interchange and display mechanism, using Unicode

1st Edition

جمهوری اسلامی ایران

موسسه استاندارد و تحقیقات صنعتی ایران

شمارهٔ استاندارد ایران

6219

فناوري اطلاعات –
تبادل و شيوهي نمايش اطلاعات فارسي براساس يوني كد

چاپ اول

آشنايي با موسسه استاندارد و تحقيقات صنعتي ايران

موسسه استاندارد و تحقيقات صنعتي ايران به موجب قانون، تنها مرجع رسمي كشور است كه عهده دار وظيفه تعيين، تدوين و نشر استانداردهاي ملي(رسمي) ميباشد.
تدوين استاندارد در رشتههاي مختلف توسط كميسيونهاي فني مركب از كارشناسان موسسه ، صاحبنظران مراكز و موسسات علمي، پژوهشي، توليدي و اقتصادي آگاه و مرتبط با موضوع صورت ميگيرد. سعي بر اين است كه استانداردهاي ملي، درجهت مطلوبيتها و مصالح ملي و با توجه به شرايط توليدي، فني و فن آوري حاصل از مشاركت آگاهانه و منصفانه صاحبان حق و نفع شامل: توليد كنندگان، مصرف كنندگان، بازرگانان، مراكز علمي و تخصصي و نهادها و سازمانهاي دولتي باشد. پيش نويس استانداردهاي ملي جهت نظرخواهي براي مراجع ذينفع و اعضاي كميسيونهاي فني مربوط ارسال ميشود. و پس از دريافت نظرات و پيشنهادها در كميته ملي مرتبط با آن رشته طرح و درصورت تصويب به عنوان استاندارد ملي(رسمي) چاپ و منتشر ميشود.
پيشنويس استانداردهايي كه توسط موسسات و سازمانهاي علاقمند و ذيصلاح و با رعايت ضوابط تعيين شده تهيه ميشود نيز پس از طرح و بررسي در كميته ملي مربوط و درصورت تصويب، به عنوان استاندارد ملي چاپ و منتشر ميگردد. بدين ترتيب استاندارهايي ملي تلقي ميشود كه بر اساس مفاد مندرج در استاندارد ملي شماره «5» تدوين و در كميته ملي مربوط كه توسط موسسه تشكيل ميگردد به تصويب رسيده باشد.
موسسه استاندارد و تحقيقات صنعتي ايران از اعضاء اصلي سازمان بينالمللي استاندارد ميباشد كه در تدوين استانداردهاي ملي ضمن توجه به شرايط كلي و نيازمنديهاي خاص كشور، از آخرين پيشرفتهاي علمي، فني و صنعتي جهان و استانداردهاي بينالمللي استفاده مينمايد.

56.0

Firefox Release

September 28, 2017

Version 56.0, first offered to Release channel users on September 28, 2017

Today's release gives Firefox users a better experience with features like Firefox Screenshots, Send Tabs, and more control over the browser with an improved (and searchable) preferences section. It also includes incremental performance improvements that move us closer to our biggest release of the year, coming in November.

We'd like to extend a special thank you to all of the new Mozillians who contributed to this release of Firefox!

new

Launched Firefox Screenshots, a feature that lets users take, save, and share screenshots without leaving the browser

Added support for address form autofill (en-US only)

Updated Preferences

- Added search tool so users can find a specific setting quickly
- Reorganized preferences so users can more easily scan settings
- Rewrote descriptions so users can better understand choices and how they affect browsing
- Revised data collection choices so they align with updated Privacy Notice and data collection strategy

Media opened in a background tab will not play until the tab is selected

Improved Send Tabs feature of Sync for iOS and Android, and Send Tabs can be discovered even by users without a Firefox Account

fixed

Various security fixes

changed

Replaced character encoding converters with a new Encoding Standard-compliant implementation written in Rust

README.md

encoding_rs

build passing crates.io v0.7.1 docs 0.7.1 license Apache 2 / MIT

encoding_rs an implementation of the (non-JavaScript parts of) the Encoding Standard written in Rust and used in Gecko (starting with Firefox 56).

Functionality

Due to the Gecko use case, encoding_rs supports decoding to and encoding from UTF-16 in addition to supporting the usual Rust use case of decoding to and encoding from UTF-8. Additionally, the API has been designed to be FFI-friendly to accommodate the C++ side of Gecko.

Specifically, encoding_rs does the following:

- Decodes a stream of bytes in an Encoding Standard-defined character encoding into valid aligned native-endian in-RAM UTF-16 (units of `u16` / `char16_t`).
- Encodes a stream of potentially-invalid aligned native-endian in-RAM UTF-16 (units of `u16` / `char16_t`) into a sequence of bytes in an Encoding Standard-defined character encoding as if the lone surrogates had been replaced with the REPLACEMENT CHARACTER before performing the encode. (Gecko's UTF-16 is potentially invalid.)
- Decodes a stream of bytes in an Encoding Standard-defined character encoding into valid UTF-8.
- Encodes a stream of valid UTF-8 into a sequence of bytes in an Encoding Standard-defined character encoding. (Rust's UTF-8 is guaranteed-valid.)
- Does the above in streaming (input and output split across multiple buffers) and non-streaming (whole input in a single buffer and whole output in a single buffer) variants.
- Avoids copying (borrows) when possible in the non-streaming cases when decoding to or encoding from UTF-8.
- Resolves textual labels that identify character encodings in protocol text into type-safe objects representing the those encodings conceptually.
- Maps the type-safe encoding objects onto strings suitable for returning from `document.characterSet` .
- Validates UTF-8 (in common instruction set scenarios a bit faster for Web workloads than the standard library; hopefully will get upstreamed some day) and ASCII.

Licensing

Please see the file named COPYRIGHT.

API Documentation

Encoding

Living Standard — Last Updated 2 October 2017

Participate:

GitHub whatwg/encoding (file an issue, open issues)
IRC: #whatwg on Freenode

Commits:

GitHub whatwg/encoding/commits
Snapshot as of this commit
@encodings

Tests:

web-platform-tests encoding/ (ongoing work)

Translation (non-normative):

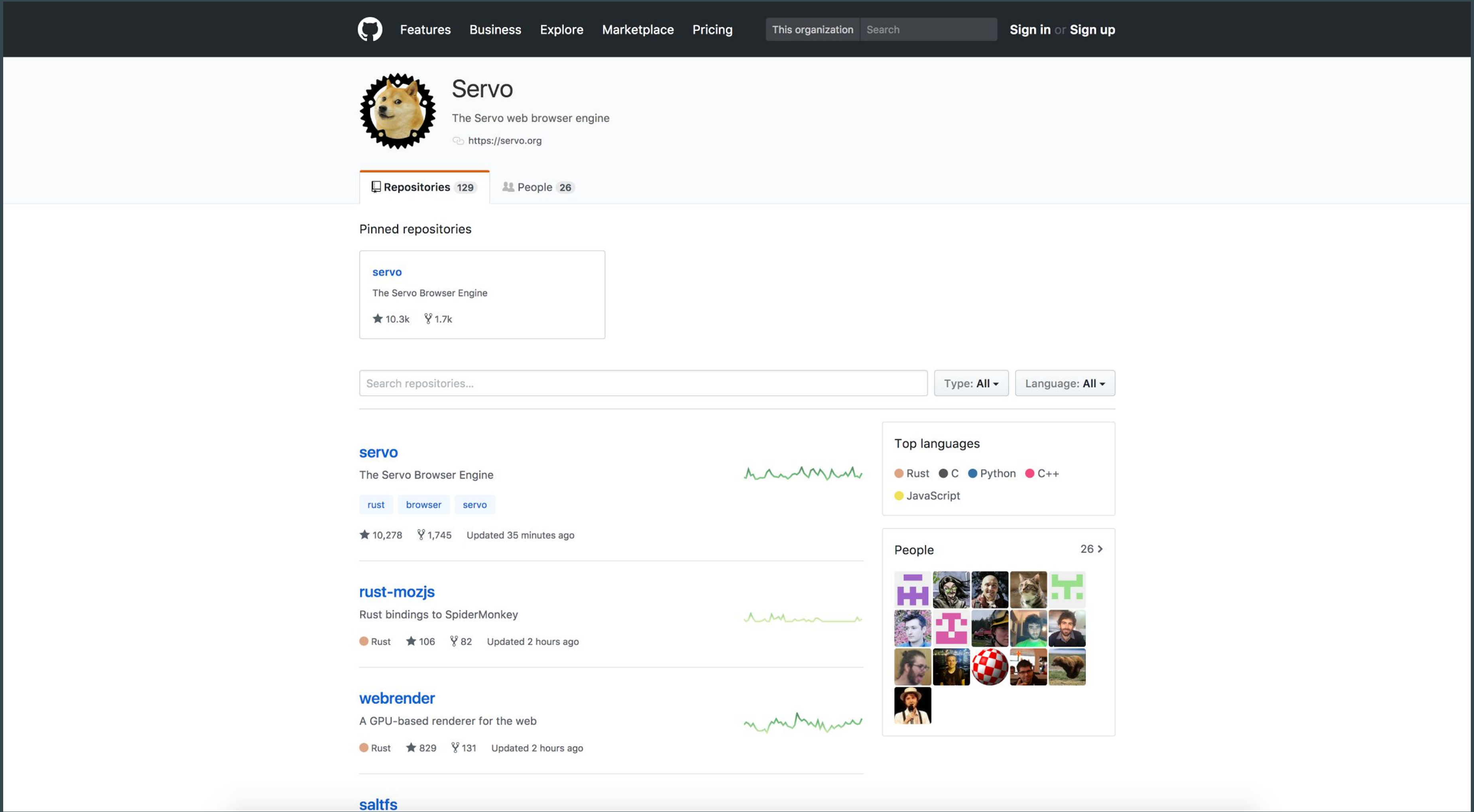
日本語

Abstract

The Encoding Standard defines encodings and their JavaScript API.

Table of Contents

File an issue about the selected text


Features
Business
Explore
Marketplace
Pricing
This organization
Search
Sign in or Sign up
Servo
The Servo web browser engine
https://servo.org
Repositories 129
People 26
Pinned repositories
servo
The Servo Browser Engine
10.3k
1.7k
Search repositories...
Type: All
Language: All
servo
The Servo Browser Engine
rust
browser
servo
10,278
1,745
Updated 35 minutes ago
rust-mozjs
Rust bindings to SpiderMonkey
Rust
106
82
Updated 2 hours ago
webrender
A GPU-based renderer for the web
Rust
829
131
Updated 2 hours ago
saltfs
Top languages
Rust
C
Python
C++
JavaScript
People
26

Fork me on GitHub

Documentation Install Community Contribute

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

See who's using Rust.

Install Rust 1.21.0

October 12, 2017

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Run

```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                     "Ciao", "こんにちは", "안녕하세요",
                     "Cześć", "Olá", "Здравствуйте",
                     "Chào bạn", "您好", "Hallo"];

    for (num, greeting) in greetings.iter().enumerate() {
        print!("{} : ", greeting);
        match num {
            0 =>  println!("This code is editable and runnabl
            1 =>  println!("¡Este código es editable y ejecut
            2 =>  println!("Ce code est modifiable et exécuta
            3 =>  println!("Questo codice è modificabile ed e
            4 =>  println!("このコードは編集して実行出来ます
            5 =>  println!("여기에서 코드를 수정하고 실행할
            6 =>  println!("Ten kod można edytować oraz uruch
            7 =>  println!("Este código é editável e executáv
            8 =>  println!("Этот код можно отредактировать и
            9 =>  println!("Bạn có thể edit và run code trực
            10 => println!("这段代码是可以编辑并且能够运行的
            11 => println!("Dieser Code kann bearbeitet und a
            _ =>  {},
        }
    }
}
```

More examples

Our site in other languages: **Deutsch**, **English**, **Español**, **Français**, **Bahasa Indonesia**, **Italiano**, **日本語**, **한국어**, **Polski**, **Português**, **Русский**, **Tiếng việt**, **简体中文**

Cargo, Rust's Package Manager

Installing

Install Stable Rust and Cargo

The easiest way to get Cargo is to get the current stable release of Rust by using the `rustup` script:

```
$ curl -sSf https://static.rust-lang.org/rustup.sh | sh
```

After this, you can use the `rustup` command to also install `beta` or `nightly` channels for Rust and Cargo.

Install Nightly Cargo

To install just Cargo, the current recommended installation method is through the official nightly builds. Note that Cargo will also require that Rust is already installed on the system.

Platform	64-bit	32-bit
Linux binaries	tar.gz	tar.gz
MacOS binaries	tar.gz	tar.gz
Windows binaries	tar.gz	tar.gz

Build and Install Cargo from Source

Alternatively, you can build Cargo from source.

Let's get started

To start a new project with Cargo, use `cargo new`:

```
$ cargo new hello_world --bin
```

We're passing `--bin` because we're making a binary program: if we were making a library, we'd leave it off.

123 lines (81 sloc) | 5.86 KB

Raw Blame History

Unicode and Rust

The Unicode Standard, and related specifications, are a complex system with interdependent terms and properties. Here's a summary for working with Unicode when programming in Rust.

Basic Unicode Concepts

- **Unicode Abstract Characters** are abstract units of information used for the organization, control, or representation of textual data.
- **Unicode Code Points** are integer values in the **Unicode codespace**: numbers between `0` (zero) and `0x10_FFFF`, inclusive.
- **Unicode Scalar Values** are integer values in a subset of *Unicode Code Points*: the *Unicode codespace* excluding high-surrogate and low-surrogate code points: `U+D800..U+DFFF`, inclusive.
- **Unicode Encoded Characters** are *Unicode Scalar Values* assigned to a *Unicode Abstract Characters* by the Unicode Standard. Some *Unicode Abstract Characters* are represented with a sequence of **Unicode Encoded Characters**.

Unicode Scalar Values marked as *noncharacters* or *reserved* (a.k.a *unassigned*) are not considered *Unicode Encoded Characters*. Therefore, *Unicode Scalar Values* can have one of the following *assignment* statuses:

- **assigned character**, code points that are marked to be an *Encoded Character*,

  Chatacter*, or

- **unassigned** or **reserved**, code points that can become an *Encoded Character* in the future.

In contrast to Unicode Code Points and Unicode Scalar Values (which are sets of numbers written in stone), the set of Unicode Encoded Characters (a subset of Scalar Values) expands with every version of Unicode. Figure below shows the number of Unicode Assigned Characters over time, from 1991 to 2017.

Primitive Type char

1.0.0 [−]

[−] A character type.

The `char` type represents a single character. More specifically, since 'character' isn't a well-defined concept in Unicode, `char` is a 'Unicode scalar value', which is similar to, but not the same as, a 'Unicode code point'.

This documentation describes a number of methods and trait implementations on the `char` type. For technical reasons, there is additional, separate documentation in the `std::char` module as well.

Representation

`char` is always four bytes in size. This is a different representation than a given character would have as part of a `String`. For example:

```
let v = vec!['h', 'e', 'l', 'l', 'o'];

// five elements times four bytes for each element
assert_eq!(20, v.len() * std::mem::size_of::<char>());

let s = String::from("hello");

// five elements times one byte per element
assert_eq!(5, s.len() * std::mem::size_of::<u8>());
```

Run

As always, remember that a human intuition for 'character' may not map to Unicode's definitions. For example, despite looking similar, the 'é' character is one Unicode code point while 'é' is two Unicode code points:

```
let mut chars = "é".chars();
// U+00e9: 'latin small letter e with acute'
assert_eq!(Some('\u{00e9}'), chars.next());
assert_eq!(None, chars.next());

let mut chars = "é".chars();
// U+0065: 'latin small letter e'
assert_eq!(Some('\u{0065}'), chars.next());
// U+0301: 'combining acute accent'
assert_eq!(Some('\u{0301}'), chars.next());
assert_eq!(None, chars.next());
```

Run

This means that the contents of the first string above *will* fit into a `char` while the contents of the second string *will not*. Trying to create a `char` literal with the contents of the second string gives an error:

Primitive Type str

Methods

Trait Implementations

std

Primitive Types

Primitive Type str

1.0.0 [−]

[−] String slices.

The `str` type, also called a 'string slice', is the most primitive string type. It is usually seen in its borrowed form, `&str`. It is also the type of string literals, `&'static str`.

Strings slices are always valid UTF-8.

This documentation describes a number of methods and trait implementations on the `str` type. For technical reasons, there is additional, separate documentation in the `std::str` module as well.

Examples

String literals are string slices:

Run

```
let hello = "Hello, world!";

// with an explicit type annotation
let hello: &'static str = "Hello, world!";
```

They are `'static` because they're stored directly in the final binary, and so will be valid for the `'static` duration.

Representation

A `&str` is made up of two components: a pointer to some bytes, and a length. You can look at these with the `as_ptr` and `len` methods:

Run

```
use std::slice;
use std::str;

let story = "Once upon a time...";

let ptr = story.as_ptr();
let len = story.len();

// story has nineteen bytes
assert_eq!(19, len);

// We can re-build a str out of ptr and len. This is all unsafe because
// we are responsible for making sure the two components are valid:
let s = unsafe {
    // First, we build a &[u8]
```

crates.io
Rust Package Registry

unicode

Browse All Crates | Docs ▾ | Log in with GitHub

Fork me on GitHub

All Crates for keyword 'unicode'

Displaying **1-10** of **55** total results

Sort by Recent Downloads ▾

unicode-xid crates.io v0.1.0

Determine whether characters have the XID_Start or XID_Continue properties according to Unicode Standard Annex #31.

Homepage Documentation Repository

All-Time: 1,667,359
Recent: 502,970

unicode-bidi crates.io v0.3.4

Implementation of the Unicode Bidirectional Algorithm

Documentation Repository

All-Time: 1,566,966
Recent: 325,470

unicode-normalization crates.io v0.1.5

This crate provides functions for normalization of Unicode strings, including Canonical and Compatible Decomposition and Recomposition, as described in Unicode Standard Annex #15.

Homepage Documentation Repository

All-Time: 1,679,200
Recent: 316,700

unicode-width crates.io v0.1.4

Determine displayed width of `char` and `str` types according to Unicode Standard Annex #11 rules.

Homepage Documentation Repository

All-Time: 956,289
Recent: 246,500

unicode-segmentation crates.io v1.2.0

This crate provides Grapheme Cluster and Word boundaries according to Unicode Standard Annex #29 rules.

Homepage Documentation Repository

All-Time: 917,916
Recent: 222,961

crates.io
Rust Package Registry

Click or press 'S' to search...

Browse All Crates | Docs ▾ | Log in with GitHub

Fork me on GitHub

All Crates for keyword 'unicode'

Displaying **11-20** of **55** total results

Sort by Recent Downloads ▾

harfbuzz-sys crates.io v0.1.15

Rust bindings to the HarfBuzz text shaping engine

All-Time: 46,434
Recent: 2,908

Documentation Repository

unicode-script crates.io v0.1.1

Look up the Unicode Script property

All-Time: 40,050
Recent: 2,128

Documentation Repository

strcursor crates.io v0.2.5

Provides a string cursor type for seeking through a string whilst respecting grapheme cluster and code point boundaries.

All-Time: 4,606
Recent: 826

Documentation Repository

unicode_categories crates.io v0.1.1

Query Unicode category membership for chars

All-Time: 2,558
Recent: 790

Documentation Repository

unic-ucd-core crates.io v0.6.0 build passing

UNIC - Unicode Character Database - Version

All-Time: 499
Recent: 298

Homepage Repository

README.md

UNIC: Unicode and Internationalization Crates for Rust

Linux build passing | Windows build passing | unicode 10.0.0 | release v0.6.0 | crates.io v0.6.0 | docs 0.6.0 | chat on gitter | chat on irc

https://github.com/behnam/rust-unic

UNIC is a project to develop components for the Rust programming language to provide high-quality and easy-to-use crates for Unicode and Internationalization data and algorithms. In other words, it's like ICU for Rust, written completely in Rust, mostly in *safe* mode, but also benifiting from performance gains of *unsafe* mode when possible.

Project Goal

The goal for UNIC is to provide access to all levels of Unicode and Internationalization functionalities, starting from Unicode character properties, to Unicode algorithms for processing text, and more advanced (locale-based) processes based on Unicode Common Locale Data Repository (CLDR).

Other standards and best practices, like IETF RFCs, are also implemented, as needed by Unicode/CLDR components, or common demand.

Project Status

At the moment, in mid-2017, UNIC is under heavy development: the API is updated frequently on `master` branch, and

Components and their Organization

UNIC *Components* have a hierarchical organization, starting from the `unic` root, containing the *major components*. Each major component, in turn, may host one or more *minor components*.

API of major components are designed for the end-users of the libraries, and are expected to be extensively documented and accompanies with code examples.

In contrast to major components, minor components act as providers of data and algorithms for the higher-level, and their API is expected to be more performing, and possibly providing multiple ways of accessing the data.

The UNIC Super-Crate

The `unic` super-crate is a collection of all UNIC (major) components, providing an easy way of access to all functionalities, when all or many are needed, instead of importing components one-by-one. This crate ensures all components imported are compatible in algorithms and consistent data-wise.

Main code examples and cross-component integration tests are implemented under this crate.

Major Components

- `unic::ucd`: Unicode Character Database. crates.io v0.6.0
- `unic::bidi`: Unicode Bidirectional Algorithm (UAX#9). crates.io v0.6.0
- `unic::normal`: Unicode Normalization Forms (UAX#15). crates.io v0.6.0
- `unic::idna`: Unicode IDNA Compatibility Processing (UTS#46). crates.io v0.6.0

Code Organization: Combined Repository

Some of the reasons to have a combined repository these components are:

- **Faster development**. Implementing new Unicode/i18n components very often depends on other (lower level) components, which in turn may need adjustments—expose new API, fix bugs, etc—that can be developed, tested and reviewed in less cycles and shorter times.
- **Implementation Integrity**. Multiple dependencies on other components mean that the components need to, to some level, agree with each other. Many Unicode algorithms, composed from smaller ones, assume that all parts of the algorithm is using the same version of Unicode data. Violation of this assumption can cause inconsistencies and hard-to-catch bugs. In a combined repository, it's possible to reach a better integrity during development, as well as with cross-component (integration) tests.
- **Pay for what you need.** Small components (basic crates), which cross-depend only on what they need, allow users to only bring in what they consume in their project.

Crate unic

Reexports
Constants

Crates

unic

Crate unic

[−] [src]

UNIC: Unicode and Internationalization Crates for Rust

The `unic` super-crate (this) is a collection of all UNIC components, providing an easy way of access to all functionalities, when all or many are needed, instead of importing components one-by-one, and ensuring all components imported are compatible in algorithms and consistent data-wise.

Components

- `ucd`: Unicode Character Database.
- `bidi`: Unicode Bidirectional Algorithm (USA#9).
- `normal`: Unicode Normalization Forms (USA#15).
- `idna`: Unicode IDNA Compatibility Processing (UTS#46).

A Basic Example

```
use unic::bidi::BidiInfo;
use unic::normal::StrNormalForm;
use unic::ucd::{Age, BidiClass, CharAge, CharBidiClass, StrBidiClass, UnicodeVersion, is_cased
use unic::ucd::normal::compose;

#[cfg_attr(rustfmt, rustfmt_skip)]
#[test]
fn test_sample() {

    // Age

    assert_eq!(Age::of('A').unwrap().actual(), UnicodeVersion { major: 1, minor: 1, micro: 0 }
    assert_eq!(Age::of('\u{A0000}'), None);
    assert_eq!(
        Age::of('\u{10FFFF}').unwrap().actual(),
        UnicodeVersion { major: 2, minor: 0, micro: 0 }
    );

    if let Some(age) = '🦊'.age() {
        assert_eq!(age.actual().major, 9);
        assert_eq!(age.actual().minor, 0);
```