# How do I convert between String, &str, Vec<u8> and &[u8]?

Asked 5 years, 11 months ago Modified 1 year, 7 months ago Viewed 20k times



A new Rustacean like me struggles with juggling these types: String, &str, Vec<u8>, &[u8].

In time, I hope to have an epiphany and suddenly get why some library calls use one or the other. Until then, I need help to map out each idiomatic transition.



Given these types:



```
let st: &str = ...;
let s: String = ...;
let u: &[u8] = ...;
let v: Vec<u8> = ...;
```

I think I have figured these out, but are they idiomatic?

```
&str -> String String::from(st)
&str -> &[u8] st.as_bytes()
String -> &str s.as_str()
\&[u8] \rightarrow \&str str::from_utf8(u)
Vec<u8> -> String String::from_utf8(v)
```

Ultimately I want a complete table of transitions for these types:

```
&str -> String
&str -> &[u8]
&str -> Vec<u8>
String -> &str
String -> &[u8]
String -> Vec<u8>
&[u8] -> &str
&[u8] -> String
&[u8] -> Vec<u8>
Vec<u8> -> &str
Vec<u8> -> String
Vec<u8> -> &[u8]
string rust type-conversion
```

Share Edit Follow

edited Apr 8, 2021 at 18:43 trent **23.1k** 7 48 86

asked Dec 8, 2016 at 8:09 Martin Algesten Martin Algesten

12.5k 3 52 77

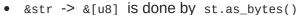
1 Answer Sorted by: Highest score (default) **\$** 



## From &str



• But I suggest you stick with one of them within a single project. The major advantage of string::from is that you can use it as an argument to a map method. So instead of x.map(|s| String::from(s)) you can often use x.map(String::from).





• &str -> Vec<u8> is a combination of &str -> &[u8] -> Vec<u8>, i.e. st.as\_bytes().to\_vec() Or st.as\_bytes().to\_owned()

## From string

- String -> &str should just be &s where coercion is available or s.as\_str() where it is not.
- String -> &[u8] is the same as &str -> &[u8]: s.as\_bytes()
- String -> Vec<u8> has a custom method: s.into\_bytes()

### From &[u8]

- &[u8] -> vec<u8> is done by u.to\_owned() or u.to\_vec(). They do the same thing, but to\_vec has the slight advantage of being unambiguous about the type it returns.
- &[u8] -> &str doesn't actually exist, that would be &[u8] -> Result<&str, Error>, provided via str::from\_utf8(u)
  - str::from\_utf8(u).unwrap() works, but you should prefer better error handling (see <u>Error handling The Result type</u>).
- &[u8] -> String is the combination of &[u8] -> Result<&str, Error> -> Result<String, Error>
  - String::from\_utf8(u).unwrap() works, but prefer better error handling (see Error handling The Result type and also Result::map.

#### From vec<u8>

- Vec<u8> -> &[u8] should be just &v where coercion is available, or as\_slice where it's not.
- Vec<u8> -> &str is the same as Vec<u8> -> &[u8] -> Result<&str, Error> i.e. str::from\_utf8(&v)
  - str::from\_utf8(&v).unwrap() works, but prefer better error handling (see Error handling The Result type)
- Vec<u8> -> String doesn't actually exist, that would be Vec<u8> -> Result<String, Error> via String::from\_utf8(v)
  - string::from\_utf8(v).unwrap() works, but prefer better error handling (see <u>Error handling</u> <u>The Result type</u>).

Coercion is available whenever the target is not generic but explicitly typed as &str or &[u8], respectively. The Rustonomicon has a chapter on coercions with more details about coercion sites.

#### tl;dr

```
&str
       -> String | String::from(s) or s.to_string() or s.to_owned()
       -> &[u8] | s.as_bytes()
      -> Vec<u8> | s.as_bytes().to_vec() or s.as_bytes().to_owned()
String -> &str | &s if possible* else s.as_str()
String -> &[u8] | s.as_bytes()
String -> Vec<u8> | s.into_bytes()
&[u8] -> &str | s.to_vec() or s.to_owned()
      -> String | std::str::from_utf8(s).unwrap(), but don't**
&[u8] -> Vec<u8> | String::from_utf8(s).unwrap(), but don't**
Vec<u8> -> &str | &s if possible* else s.as_slice()
Vec<u8> -> String | std::str::from_utf8(&s).unwrap(), but don't**
Vec<u8> -> &[u8] | String::from_utf8(s).unwrap(), but don't**
```

- \* target should have explicit type (i.e., checker can't infer that)
- \*\* handle the error properly instead

Share Edit Follow edited Aug 18, 2019 at 18:06

community wiki 12 revs, 7 users 26% Martin Algesten

OK, but how to change a single u8 variable to string? to\_string() seems to not work, since it yells "no method named as\_string found for type u8 in the current scope"... What I want is to change a small number (like 32) to string "32". – piotao Jul 29, 2020 at 21:49

@piotao for that you need to call format!("{}", 5\_u8), which returns an owned String.-rdxdkr Jun 14 at 20:36