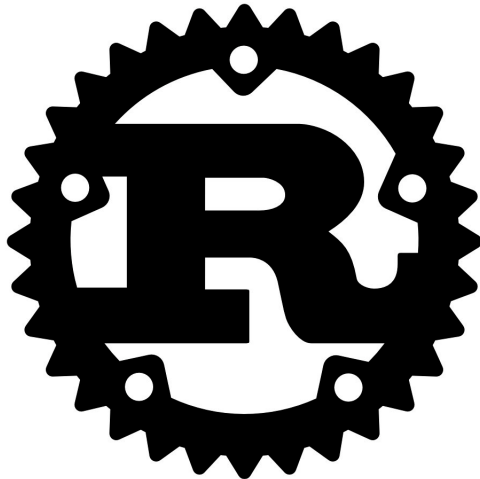

RUST CHEAT SHEET

Jaideep Ganguly

Last updated on August 24, 2022



**The Rust
Programming
Language**

Contents

Contents	2
1 Introduction	4
1.1 Definitions	4
1.2 Installation	4
1.3 Cargo - Rust Package Manager	4
1.4 Data Types	5
1.5 Program Structure	5
2 Ownership, Shadowing, Referencing & Lifetime	6
2.1 Ownership & Shadowing	6
2.2 Lifetime	6
2.3 String Functions <u>Struct std::string::String</u>	7
3 Flow	8
4 Loop	8
5 Data Structures	9
5.1 Tuple, Array, Slice	9
5.2 Struct	9
6 Trait	10
6.1 Traitbound	10
7 Generic	11
8 Enum	12
9 Collections	13
9.1 Vec	13
9.2 HashMap	13
10 Closure	14
11 Error Handling	14
12 Smart Pointers	15
12.1 Deref	15
12.2 Drop	15
13 Concurrency	16
13.1 Thread	16
14 IO	19
15 JSON	20
16 Database	21
17 <u>Tonic - Rust implementation of gRPC</u>	22
17.1 <u>Java Client for Rust Service</u>	24
Bibliography	24

Listings

1	Installation	4
2	Creating new package with Cargo	4
3	Example Cargo Listing	4
4	Data Types	5
5	Suppress Compiler Warnings	5
6	Function Structure	5
7	Function Invocation	5
8	Ownership & Shadowing	6
9	Lifetime	6
10	String Functions	7
11	Flow	8
12	Loop	8
13	Tuple, Array, Slice	9
14	Struct	9
15	Trait	10
16	Traitbound	10
17	Trait Example	11
18	Generic	11
19	Enum	12
20	Enum Match	12
21	Some	12
22	Result	12
23	Vec	13
24	HashMap	13
25	Closure	14
26	Error Handling	14
27	Deref	15
28	Drop	15
29	Thread	16
30	Mutex	16
31	Message Passing	17
32	Long running fh 1	18
33	Long running fh 2	18
34	Invoking Future	18
35	IO	19
36	JSON	20
37	DB Server	21
38	Creating new package with Cargo	22
39	proto/helloworld.proto	22
40	Cargo.toml	22
41	build.rs (at root of project)	23
42	server.rs	23
43	client.rs	23

1 Introduction

1.1 Definitions

- ① Packages - A Cargo feature that lets you build, test, and share crates.
- ② Crates - A tree of modules that produces a library or executable.
- ③ Modules and use - Let you control the organization, scope, and privacy of paths.
- ④ Paths - A way of naming an item, such as a struct, function, or module.

1.2 Installation

```
1 curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh # Install
2 rustup update          # Update
3 rustup self uninstall  # Uninstall
4 rustc --version        # Version
```

Listing 1: Installation

1.3 Cargo - Rust Package Manager

```
1 cargo new my-project      # Creating a new package with a binary crate
2 cargo new my-project --lib # Creating a new package with a library crate
```

Listing 2: Creating new package with Cargo

```
1 [package]
2 name = "tpl"
3 version = "0.1.0"
4 authors = ["Jaideep Ganguly <ganguly.jaideep@gmail.com>"]
5 edition = "2021"
6
7 # See more keys and their definitions at
8   ↪ https://doc.rust-lang.org/cargo/reference/manifest.html
9
10 [dependencies]
11 futures = "0.3.0"
12 tokio = {version = "0.2.*", features = ["full"]}
13 reqwest = {version = "0.10.0-alpha.1", features = ["json"]}
14 serde_json = "1.0"
15 rand = "0.5.5"
16 mysql = "*"
17
```

Listing 3: Example Cargo Listing

1.4 Data Types

```
1 bool // Boolean
2 u8, u16, u32, u64, u128 // Unsigned integers
3 i8, i16, i32, i64, i128 // Signed integers
4 f32, f64 // Floating point numbers
5 usize // Unsigned integer, platform specific
6 isize // Signed integer, platform specific
7 char // Unicode scalar value
8 &str // String literal, aka string slice, is a sequence of Unicode
    ↪ characters, value is known at compile time, static by default,
    ↪ guaranteed to be valid for the duration of the entire program.
9 String // Growable Mutable Collection. String object is used to represent
    ↪ string values that are provided at runtime, allocated in the heap.
```

Listing 4: Data Types

1.5 Program Structure

```
2 #![allow(unused_assignments)]
3 #![allow(unused_imports)]
4 #![allow(unused_variables)]
5 #![allow(dead_code)]
```

Listing 5: Suppress Compiler Warnings

```
10 pub fn fn_ex(x:i32) -> i32 { // returns i32.
11                               // The keyword pub makes any module,
12     println!("{}",x);         // function, or data structure
13     return x+1;               // accessible from inside of external
    ↪ modules.
14 }
```

Listing 6: Function Structure

```
1 mod mod_tpl; // functions in mod_tpl.rs ; Alternately, functions
2 mod_tpl::typ_fn(33); // in mod.rs under directory mod_tpl
```

Listing 7: Function Invocation

2 Ownership, Shadowing, Referencing & Lifetime

2.1 Ownership & Shadowing

```
22 pub fn variable_ex() {
23     let b:bool = true;           // let introduces a variable into the
24     println!("b={}",b);         // current scope
25     let i:i32 = 300;
26
27     let x = 101.25;              // type inference
28
29     let mut f:f64 = 3.14;        // Variables are immutable by default,
30     f = 3.14159;                // mut keyword makes it mutable.
31
32     type NanoSecond = u64;      // type alias
33     const MAX_PTS: u32 = 100_000; // constant
34     static mut COUNTER: u32 = 0; // stored in dedicated memory location
35
36     let b = "Hello";           // Shadowing allows you to re-declare
37     println!("b={}",b);        // a variable in the same scope, using the same
    ↪ name. The re-declared variable differs from the original by having a
    ↪ different type. This is especially useful for casting data from one
    ↪ type into another.
38
39     let mut s:&str = "Jaideep";
40     s = "Ganguly";
41     let s = String::from("Hi Jaideep!");
42 }
```

Listing 8: Ownership & Shadowing

2.2 Lifetime

```
46 pub fn lifetime_ex<'t>(x: &'t str, y: &'t str) -> &'t str {
47     if x.bytes().len() > y.bytes().len() {
48         x
49     } else {
50         y
51     }
52 }
```

Listing 9: Lifetime

2.3 String Functions Struct std::string::String

```
61 pub fn string_ex(mut s:String) {
62     s = String::from("Jaideep Ganguly");
63
64     let mut litstr = "Hello Jaideep!"; // convert literal string to String
65     s = litstr.to_string();
66     s = s.replace("Jaideep","Mone");    // not in place, returns new String
67     litstr = s.as_str();                // convert to literal
68
69     s.push('G');                        // append a char
70     s.push_str("anguly");               // append a slice
71     println!("{}",s.len());           // length
72     println!("{}",s.trim());           // remove leading and trailing spaces
73     let s = s.clear();                 // clear
74
75     let s = "Jaideep Ganguly";
76     let v: Vec<&str> = s.split_whitespace().collect(); // split by whitesp.
77     for token in v {
78         println!("{}",token);
79     }
80     let s = "Jaideep.Ganguly"; // split
81     let x = s.split(".");
82     for token in x {
83         println!("{}",token);
84     }
85     let x = s.chars();                 // chars
86     for token in x {
87         println!("{}",token);
88     }
89     let s1 = String::from("Jaideep"); // concatenation
90     let s2 = String::from("Ganguly");
91     let mut s = s1 + &s2;
92     let s1 = String::from("Jaideep"); // concatenation
93     let s2 = String::from("Ganguly");
94     s = format!("{}", s1 - s2,s1,s2);
95     s = s.to_uppercase();               // uppercase
96     s = s.to_lowercase();              // lowercase
97 }
```

Listing 10: String Functions

3 Flow

```
105 pub fn flow(x:i32) {  
106     if x < 10 {  
107         println!("{:?}", "Less than 10");  
108     }  
109     else if x > 10 {  
110         println!("{:?}", "Greater than 10");  
111     }  
112     else {  
113         println!("{:?}", "equal to 10");  
114     }  
115 }
```

Listing 11: Flow

4 Loop

```
119 pub fn loop_ex() {  
120     for n in (1..11).step_by(2) {    // excludes 11  
121         println!("{}", n);  
122     }  
123  
124     let names = vec!["Jaideep", "Ganguly"];  
125     for name in names.iter() {  
126         println!("{}", name);  
127     }  
128  
129     let mut n = 0;  
130     while n < 11 {  
131         n += 2;  
132         print!("{}", n);  
133     }  
134 }
```

Listing 12: Loop

5 Data Structures

5.1 Tuple, Array, Slice

```
142 pub fn tup_arr_sli_ex() {
143     let tup1: (i32, f64, String) = (10, 200.32, String::from("Jai")); // tuple
144     let tup2: (i32, f64, &str) = (10, 200.32, "Ganguly");
145     println!("{:?} {:?}", tup1, tup2); // ? implements std::fmt::Display
146
147     let arr1 = [1, 2, 3, 4, 5]; // array. Must have a known length,
148     println!("{:?}", arr1); // all elements must be initialized.
149     let arr2: [i32; 3] = [0; 3]; // Length determined at compile time.
150     println!("{:?}", arr2);
151     for item in arr2.iter().enumerate() {
152         let (i, x): (usize, &i32) = item;
153         println!("array[{}] = {}", i, x);
154     }
155
156     let slice = &arr1[1..3]; // slice; length determined at runtime;
157     println!("{:?}", slice); // excludes arr1[3]
158 }
```

Listing 13: Tuple, Array, Slice

5.2 Struct

```
162 #[derive(Debug)]
163 pub struct Rect<'a> {
164     pub id: &'a str,
165     pub width: i32,
166     pub length: i32
167 }
168
169 impl<'a> Rect<'a> {
170     pub fn area(&self) -> i32 {
171         self.width * self.length
172     }
173
174     pub fn volume(&self, height: i32) -> i32 {
175         self.area() * height
176     }
177 }
178
179 pub fn struct_ex() {
180     let r = Rect {id: "id100", width: 10, length: 20};
181     println!("{:?}", r);
182     println!("Area = {}", r.area());
183     println!("Volume = {}", r.volume(10));
184 }
```

Listing 14: Struct

6 Trait

```
193 pub trait TrAnimal {
194     fn eat(&self) {
195         println!("I eat grass");
196     }
197 }
198
199 pub struct Herbivore;
200
201 impl TrAnimal for Herbivore{
202     fn eat(&self) {
203         println!("I eat plants");
204     }
205 }
206
207 pub struct Carnivore;
208
209 impl TrAnimal for Carnivore {
210     fn eat(&self) {
211         println!("I eat meat");
212     }
213 }
```

Listing 15: Trait

6.1 Traitbound

```
217 pub trait TrActivity {
218     fn fly(&self);
219 }
220
221 #[derive(Debug)]
222 pub struct Eagle;
223
224 impl TrActivity for Eagle {
225     fn fly(&self) {
226         println!("{:?} is flying",&self);
227     }
228 }
229
230 pub fn activity<T: TrActivity + std::fmt::Debug>(bird: T) {
231     println!("I fly as {:?}",bird);
232 }
```

Listing 16: Traitbound

```

236 pub fn trait_ex() {
237     use TrAnimal;
238     let h = Herbivore;
239     h.eat();
240
241     let c = Carnivore;
242     c.eat();
243
244     use TrActivity;
245     let eagle = Eagle;
246     eagle.fly();
247     activity(eagle);
248
249     /*****
250     let hen = mod_tpl::Hen; // Compile Error because hen does
251     mod_tpl::activity(hen); // not implement TrActivity trait
252     *****/
253 }

```

Listing 17: Trait Example

7 Generic

```

261 pub fn generic_ex() {
262     struct Data<T> {
263         value:T,
264     }
265
266     let t:Data<i32> = Data{value:350}; // i32
267     println!("value is :{}",t.value);
268
269     let t2:Data<String> = Data{value:"Tom".to_string()}; // String
270     println!("value is :{}",t2.value);
271     // end main_generic
272 }

```

Listing 18: Generic

8 Enum

```
280 #[derive(Debug)]
281 pub enum Command {
282     Quit,
283     Move { x: i32, y: i32 },
284     Speak(String),
285     ChangeBGColor(i32, i32, i32),
286 }
```

Listing 19: Enum

```
290 pub fn enum_ex() {
291     let msg = Command::ChangeBGColor(10, 20, 30);
292
293     match msg {
294         Command::Quit => {
295             println!("{:?}",msg);
296         },
297         Command::ChangeBGColor(r,g,b) => {
298             println!("{}", r, g, b);
299         },
300         _ => {
301             println!("{:?}", "Non ChangeBGColor");
302         }
303         _ => {
304             println!("{:?}", "Non Quit");
305         }
306     }
307
308     if let msg = Command::ChangeBGColor(10,20,30) {
309         println!("{:?}", "ok");
310     }
311 }
```

Listing 20: Enum Match

```
315 enum Option<T> {
316     Some(T), // used to return a value
317     None     // used to indicate null, Rust does not support null
318 }
```

Listing 21: Some

```
323 enum Result<T,E> {
324     OK(T),
325     Err(E)
326 }
```

Listing 22: Result

9 Collections

9.1 Vec

```
335 pub fn vec_ex() {
336
337     let mut v = vec!["Hello","how","are","you"]; // create a vector
338     v.push("today"); // push
339     v.pop(); // pop
340     v.insert(4, "sir"); // insert a value in a particular position
341     v.remove(0); // remove a value by its index
342
343     let index = v.iter().position(|x| x == &"sir").unwrap(); // rm value
344     v.remove(index);
345
346     for i in &v { // iterate, & required because v is moved due to
347         ↪ implicit call to .into_iter()
348         println!("{}", i);
349     }
350
351     for (i, elem) in v.iter().enumerate() {
352         println!("Element at position {}: {:?}", i, elem);
353     }
```

Listing 23: Vec

9.2 HashMap

```
357 use std::collections::HashMap;
358 pub fn hashmap_ex() {
359     let mut hm: HashMap<String,String> = HashMap::new();
360     hm.insert("MA".to_string(),"Massachusetts".to_string());
361     hm.insert("NY".to_string(),"New York".to_string());
362     hm.insert("CA".to_string(),"California".to_string());
363
364     for (key, val) in hm.iter() {
365         println!("key: {} val: {}", key, val);
366     }
367
368     *hm.get_mut("MA").unwrap() = "MASSACHUSETTS".to_string();
369
370     hm.remove("CA");
371     for (key, val) in hm.iter() {
372         println!("key: {} val: {}", key, val);
373     }
374     println!("{:?}", hm.len());
375 }
```

Listing 24: HashMap

10 Closure

```
385 pub fn closure_ex1() {
386     use std::thread;
387     use std::time::Duration;
388     let some_closure = |number: u32| -> u32 {
389         println!("calculating ...");
390         thread::sleep(Duration::from_secs(3));
391         number + 1
392     };
393 }
394
395 pub fn closure_ex2(x: i32) -> i32 {
396     let y = 3;
397     let add = |x| {
398         x + y
399     };
400
401     let result = receive_closure(add, x);
402     result
403 }
404
405 fn receive_closure<F>(f: F, x: i32) -> i32
406     where F: Fn(i32) -> i32 {
407     f(x) // as i32
408 }
```

Listing 25: Closure

11 Error Handling

```
417 pub fn error_ex() {
418     let x = 5;
419     if (x > 10) {
420         panic!("I am panicking, can't proceed any further");
421     }
422     println!("I won't print this");
423
424     // let f = File::open("/xyz/file.txt").expect("File not found");
425     let f = File::open("/Users/jaideep.ganguly/rust/src/inp.txt");
426     match f {
427         Ok(f) => {
428             println!("file: {:?}",f);
429         },
430         Err(e) => {
431             println!("file not found{:?}",e); // handled error
432         }
433     }
434     println!("I will print this");
435 }
```

Listing 26: Error Handling

12 Smart Pointers

12.1 Deref

```
444 pub fn deref_ex() {
445     let x = 5;
446     let y = Box::new(x);
447     println!("{:?}", "Checking");
448     assert_eq!(5,x);    // will panic if false
449     assert_eq!(5,*y);
450
451
452     #[derive(Debug)]
453     struct MyBox<T> { // same as: struct MyBox<T>(T);
454         a: T
455     }
456
457     use std::ops::Deref;
458     impl<T> Deref for MyBox<T> {
459         type Target = T;
460
461         fn deref(&self) -> &T {
462             &self.a
463         }
464     }
465
466     let x = MyBox{a:100};
467     println!("{:?}",x);           // output: MyBox { a: 100 }
468     println!("{}",*(x.deref())); // output: 100
469 }
```

Listing 27: Deref

12.2 Drop

```
473 pub fn drop_ex() {
474     let x = mysmaptr{ data : String::from("Hello") };
475     println!("struct mysmaptr with data {}", x.data);
476
477     struct mysmaptr {
478         data: String
479     }
480
481     impl Drop for mysmaptr {
482         fn drop(&mut self) {
483             println!("Dropping struct mysmaptr with data {}", self.data);
484         }
485     }
486 }
```

Listing 28: Drop

13 Concurrency

13.1 Thread

```
494 use std::thread;
495 use std::sync::{Arc, Mutex};
496 use std::time::{Duration, Instant};
497 use std::process;
498 use std::sync::mpsc;
499 use futures::future;
500 use futures::join;
501 use futures::try_join;
502 use tokio::macros::support::Future;
503
504 pub fn thread_ex() {
505     let handle = thread::spawn( || {
506         for i in 1..10 {
507             println!("Hello # {} from the spawned thread!", i);
508             thread::sleep(Duration::from_millis(1));
509         }
510     });
511
512     for i in 1..5 {
513         println!("Hi # {} from the main thread!", i);
514         ↪ thread::sleep(Duration::from_millis(1));
515     }
516
517     handle.join().unwrap();
518 }
```

Listing 29: Thread

```
520 pub fn mutex_ex() {
521     let counter = Arc::new(Mutex::new(100)); // atomic ref count
522     let mut handles = vec![];                // stores refs to threads
523
524     for _ in 0..10 {                          // spawn 10 threads
525         let counter = Arc::clone(&counter); // clone the arc
526         let handle = thread::spawn( move || { // move closure
527             let mut num = counter.lock().unwrap();
528             *num += 1;
529         });
530         handles.push(handle);
531     }
532
533     for handle in handles {                    // join the threads
534         handle.join().unwrap();
535     }
536
537     println!("Result: {}", *counter.lock().unwrap());
538 }
```

Listing 30: Mutex


```

542 pub fn msgpass_ex() {
543     // Channel to send and receive messages between concurrent sections of
    ↪ code; has two halves, a transmitter and a receiver.
544
545     let (tx, rx) = mpsc::channel(); // multiple producer, 1 consumer
546     let tx2 = mpsc::Sender::clone(&tx); // clone a second producer
547
548     // spawn a thread, move the transmitter into the closure
549     // spawned thread will now own the transmitter
550     thread::spawn( move || {
551         let vals = vec![
552             String::from("Hello"),
553             String::from("from"),
554             String::from("thread-1"),
555         ];
556
557         for val in vals {
558             tx.send(val).unwrap();
559             thread::sleep(Duration::from_secs(1));
560         }
561     });
562
563     thread::spawn( move || { // same comments as above
564         let vals = vec![
565             String::from("Hi"),
566             String::from("there"),
567             String::from("thread-2"),
568         ];
569
570         for val in vals {
571             tx2.send(val).unwrap();
572             thread::sleep(Duration::from_secs(1));
573         }
574     });
575
576     // receive the result, timeout beyond 1 sec
577     let result = rx.recv_timeout(Duration::from_millis(1000));
578
579     match result {
580         Err(e) => {
581             println!("{:?}", e);
582             process::exit(0);
583         },
584         Ok(x) => {
585             for received in rx {
586                 println!("Got: {}", received);
587             }
588         }
589     }

```

```
590 }
```

Listing 31: Message Passing

```
595 pub async fn long_running_fn_1(x: &mut i32) -> i32 {
596     thread::sleep(Duration::from_secs(1));
597     *x = *x + 1;
598     thread::sleep(Duration::from_secs(1));
599     *x
600 }
```

Listing 32: Long running fh 1

```
604 pub async fn long_running_fn_2() -> i32 {
605     thread::sleep(Duration::from_secs(4));
606     42
607 }
```

Listing 33: Long running fh 2

```
58     let t1 = Instant::now();
59     let mut x1 = 100;
60     let r1 = mod_tpl::long_running_fn_1(&mut x1).await; // Sequential exec.
61     let r2 = mod_tpl::long_running_fn_2().await;
62     let t2 = Instant::now();
63     println!("{}", r1, r2, t2 - t1);
64
65     let tasks = vec![ // Concurrent execution
66         tokio::spawn(async move { mod_tpl::long_running_fn_1(&mut
↪ x1).await }),
67         tokio::spawn(async move { mod_tpl::long_running_fn_2().await }),
68     ];
69
70     let t1 = Instant::now();
71     let r = futures::future::join_all(tasks).await; // join the tasks
72     let t2 = Instant::now();
73     println!("{}", r, t2 - t1);
```

Listing 34: Invoking Future

```

643 use std::fs::File;
644 use std::io::Write;
645 use std::io::Read;
646 use std::fs::OpenOptions;
647 use std::fs;
648
649 pub fn std_inp() {
650     let mut line = String::new();
651     println!("Please enter your name:");
652     let nb = std::io::stdin().read_line(&mut line).unwrap();
653     println!("Hi {}", line);
654     println!("# of bytes read , {}", nb);
655 }
656
657 pub fn std_out() {
658     let b1 = std::io::stdout()
659         .write("Hi ".as_bytes()).unwrap();
660     let b2 = std::io::stdout()
661         .write(String::from("There\n").as_bytes()).unwrap();
662     std::io::stdout().
663         write(format!("#bytes written {}", (b1+b2))
664             .as_bytes()).unwrap();
665 }
666
667 pub fn cl_arg() {
668     let cmd_line = std::env::args();
669     println!("# of command line arguments:{}",cmd_line.len());
670     for arg in cmd_line {
671         println!("{}",arg);
672     }
673 }
674
675 pub fn file_read(filename: &str){
676     let mut file = std::fs::File::open(filename).unwrap();
677     let mut contents = String::new();
678     file.read_to_string(&mut contents).unwrap();
679     print!("{}", contents);
680 }
681
682 pub fn file_write(filename: &str, s: &str) {
683     let mut file = std::fs::File::create(filename)
684         .expect("Create failed");
685     file.write_all(s.as_bytes())
686         .expect("write failed");
687     println!("Write completed" );
688 }
689
690 pub fn file_append(filename: &str, s: &str) {
691     let mut file = OpenOptions::new()

```

```

691     .append(true).open(filename)
692     .expect("Failed to open file");
693     file.write_all(s.as_bytes()).expect("write failure");
694     println!("Appended file {}", filename);
695 }
696
697 pub fn file_copy(src: &str, des: &str) {
698     let mut file_inp = std::fs::File::open(src).unwrap();
699     let mut file_out = std::fs::File::create(des).unwrap();
700     let mut buffer = [0u8; 4096];
701     loop {
702         let nbytes = file_inp.read(&mut buffer).unwrap();
703         file_out.write(&buffer[..nbytes]).unwrap();
704         if nbytes < buffer.len() {
705             break;
706         }
707     }
708 }
709
710 pub fn file_delete(filename: &str) {
711     fs::remove_file(filename).expect("Unable to delete file");
712     println!("Deleted file {}", filename);
713 }

```

Listing 35: IO

15 JSON

```

615 use serde::{Deserialize, Serialize};
616
617 #[derive(Debug, Deserialize, Serialize)]
618 struct Person {
619     name: String,
620     age: usize,
621     verified: bool,
622 }
623
624 pub fn json_ex() {
625     let json = r#"
626         {
627             "name": "George",
628             "age": 27,
629             "verified": false
630         }
631     "#;
632     let p: Person = serde_json::from_str(json).unwrap(); // JSON to Struct
633     println!("{:?}", p);
634     let j = serde_json::to_string(&p); // Struct to JSON
635     println!("{:?}", j.unwrap());
636 }

```

Listing 36: JSON

```

720 use mysql::*;
721 use mysql::prelude::*;
722 use chrono::prelude::*; //For date and time
723
724 #[derive(Debug, PartialEq, Eq)]
725 struct Tab {
726     cat: String,
727     tsk: String
728 }
729
730 pub fn dbs() {
731
732     let url = "mysql://root:root@localhost:3306/pgm";
733     let opts:Opts = Opts::from_url(url).unwrap();
734     let pool = Pool::new(opts).unwrap();
735     let mut conn = pool.get_conn().unwrap();
736
737
738     let selected_tab = conn.query_map( // Select
739         "SELECT cat, tsk FROM pgm.pgm",
740         | (cat, tsk) | {
741             Tab { cat,tsk }
742         },
743     ).unwrap();
744
745     for r in selected_tab.iter() {
746         println!("{}", r.cat, r.tsk);
747     }
748
749
750     // let rows = vec![
751     //     Tab { co1: "hi".to_string(), co2: 2, },
752     //     Tab { co1: "hello".to_string(), co2: 4, },
753     // ];
754
755     // conn.exec_batch( // insert
756     //     r"INSERT INTO tpl.tab (catt, tsk)
757     //     VALUES (:cat, :tsk)",
758     //     rows.iter().map(|p| params! {
759     //         "cat" => String::from(&p.cat),
760     //         "tsk" => p.tsk,
761     //     })
762     // ).unwrap();
763
764
765 }

```

Listing 37: DB Server

17 Tonic - Rust implementation of gRPC

```
1 cargo new tonic_ex # A tonic example
2 cd tonic_ex
3 mkdir proto
4 touch proto/helloworld.proto
```

Listing 38: Creating new package with Cargo

```
1 syntax = "proto3";
2 package helloworld;
3
4 service Greeter {
5     rpc SayHello (HelloRequest) returns (HelloReply);
6 }
7
8 message HelloRequest {
9     string name = 1;
10 }
11
12 message HelloReply {
13     string message = 1;
14 }
```

Listing 39: proto/helloworld.proto

```
1 [package]
2 name = "tonic_ex"
3 version = "0.1.0"
4 edition = "2021"
5
6 # See more keys and their definitions at
7   ↪ https://doc.rust-lang.org/cargo/reference/manifest.html
8
9 [[bin]] # Bin to run the HelloWorld gRPC server
10 name = "helloworld-server"
11 path = "src/server.rs"
12
13 [[bin]] # Bin to run the HelloWorld gRPC client
14 name = "helloworld-client"
15 path = "src/client.rs"
16
17 [dependencies]
18 tonic = "0.7"
19 prost = "0.10"
20 tokio = { version = "1.0", features = ["macros", "rt-multi-thread"] }
21
22 [build-dependencies]
23 tonic-build = "0.7"
```

Listing 40: Cargo.toml

```

1 fn main() -> Result<(), Box<dyn std::error::Error>> {
2     tonic_build::compile_protos("proto/helloworld.proto"?);
3     Ok(())
4 }

```

Listing 41: build.rs (at root of project)

```

1 use tonic::{transport::Server, Request, Response, Status};
2 use hello_world::greeter_server::{Greeter, GreeterServer};
3 use hello_world::{HelloReply, HelloRequest};
4
5 pub mod hello_world {
6     tonic::include_proto!("helloworld");
7 }
8
9 #[derive(Debug, Default)]
10 pub struct MyGreeter {}
11
12 #[tonic::async_trait]
13 impl Greeter for MyGreeter {
14     async fn say_hello(&self, request: Request<HelloRequest>,) ->
15         ↪ Result<Response<HelloReply>, Status> {
16         println!("Got a request: {:?}", request);
17
18         let reply = hello_world::HelloReply {
19             message: format!("Hello {}!",
20                             request.into_inner().name).into(),
21         };
22
23         Ok(Response::new(reply))
24     }
25 }
26
27 #[tokio::main]
28 async fn main() -> Result<(), Box<dyn std::error::Error>> {
29     let addr = "[::1]:50051".parse()?;
30     let greeter = MyGreeter::default();
31
32     Server::builder()
33         .add_service(GreeterServer::new(greeter))
34         .serve(addr)
35         .await?;
36
37     Ok(())
38 }

```

Listing 42: server.rs

```

1 use hello_world::greeter_client::GreeterClient;
2 use hello_world::HelloRequest;
3
4 pub mod hello_world {

```

```

5     tonic::include_proto!("helloworld");
6 }
7
8 #[tokio::main]
9 async fn main() -> Result<(), Box<dyn std::error::Error>> {
10     let mut client = GreeterClient::connect("http://[::1]:50051").await?;
11     let request = tonic::Request::new(HelloRequest {
12         name: "Tonic".into(),
13     });
14     let response = client.say_hello(request).await?;
15     println!("RESPONSE={:?}", response);
16     Ok(())
17 }

```

Listing 43: client.rs

17.1 Java Client for Rust Service

[Java Client - Rust Server connectivity using Tonic](#)

Bibliography

- [1] A Book on Rust. [Moving to the Rust Programming Language by Jaideep Ganguly](#)
- [2] A Slide Deck on Rust. [A Slide Deck on Rust by Jaideep Ganguly](#)