

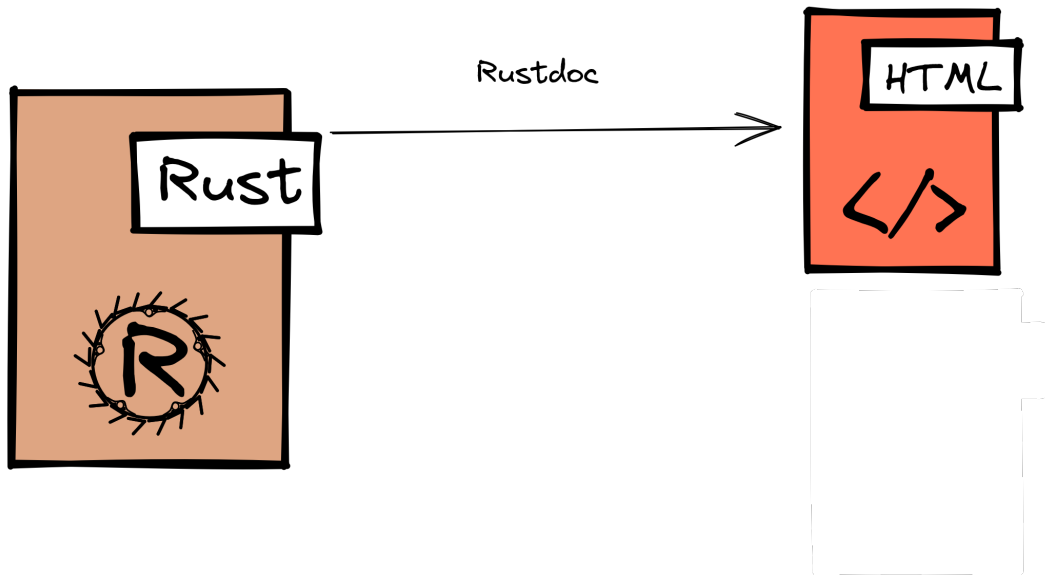
# Rustdoc JSON

[adotinthevoid.github.io/talks/rustdoc-json.pdf](https://adotinthevoid.github.io/talks/rustdoc-json.pdf)

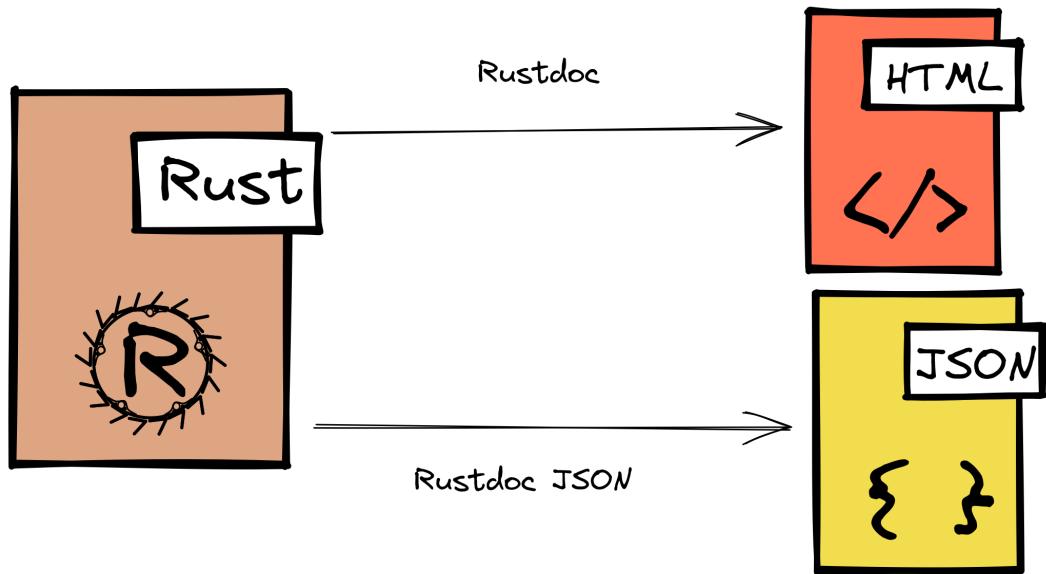
Alona Enraght-Moony

2022-10-26

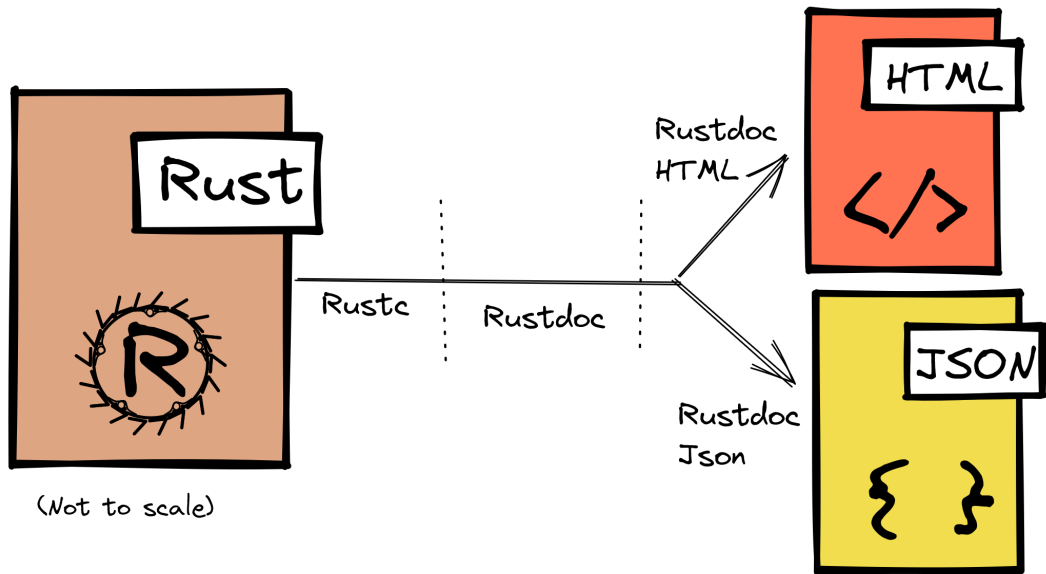
What is this?



What is this?



What is this?



Users: Roogle

# Roogle

set:libstd



```
fn (String) → &str
```

## alloc::string::String::as\_str

Extracts a string slice containing the entire String.

## Examples

Basic usage:

```
let s = String::from("foo");
```

```
assert_eq!("foo", s.as_str());
```

## Users: cargo-check-external-types

```
$ cat src/lib.rs
pub fn uh_oh(x: otherlib::ExternalStruct) {}

$ cargo +nightly-2022-07-25 check-external-types
Running rustdoc to produce json doc output...
Examining all public types...
error: Unapproved external type `otherlib::ExternalStruct` referenced in public API
1 errors emitted
$ |
```

# Users: cargo-semver-check

```
Starting 8 checks, version 0.1.0 -> 0.1.0 (no change)
PASS [ 0.000s]      major      enum_missing
FAIL [ 0.000s]      major      enum_variant_added
PASS [ 0.000s]      major      enum_variant_missing
PASS [ 0.000s]      major      struct_marked_non_exhaustive
PASS [ 0.000s]      major      struct_missing
PASS [ 0.000s]      major      struct_pub_field_missing
PASS [ 0.000s]      major      unit_struct_changed_kind
PASS [ 0.000s]      major      variant_marked_non_exhaustive
Summary [ 0.003s] 8 checks run: 7 passed, 1 failed, 0 skipped
```

--- failure enum\_variant\_added: enum variant added on exhaustive enum ---

## Description:

A publicly-visible enum without #[non\_exhaustive] has a new variant.

ref: <https://doc.rust-lang.org/cargo/reference/semver.html#enum-variant-new>

impl: [https://github.com/obi1kenobi/cargo-semver-check/tree/v0.4.1/src/queries/enum\\_variant\\_added.ron](https://github.com/obi1kenobi/cargo-semver-check/tree/v0.4.1/src/queries/enum_variant_added.ron)

## Failed in:

variant EnumWithNewVariant::NewVariant in src/test\_cases/enum\_variant\_added.rs:5

```
Final [ 0.003s] semver requires new major version: 1 major and 0 minor checks failed
```

## Just Use Rustc

```
rustup component add rust-src rustc-dev llvm-tools-preview
```



## Just Use Rustc: Imports

```
#![feature(rustc_private)]

extern crate rustc_error_codes;
extern crate rustc_errors;
extern crate rustc_hash;
extern crate rustc_hir;
extern crate rustc_interface;
extern crate rustc_session;
extern crate rustc_span;

use std::{path, process, str};

use rustc_errors::registry;
use rustc_session::config;
use rustc_span::symbol::Symbol;
```

## Just Use Rustc: Find Sysroot

```
fn main() {  
    let path: path::PathBuf = std::env::args().nth(1).unwrap().into();  
  
    let out = process::Command::new("rustc")  
        // $HOME/.rustup/toolchains/nightly-x86_64-unknown-linux-gnu  
        .arg("--print=sysroot")  
        .current_dir(".")  
        .output()  
        .unwrap();  
    let sysroot = str::from_utf8(&out.stdout).unwrap().trim();  
}
```

## Just Use Rustc: Config

```
let config = rustc_interface::Config {  
    opts: config::Options {  
        maybe_sysroot: Some(path::PathBuf::from(sysroot)),  
        ..config::Options::default()  
    },  
    registry: registry::Registry::new(&rustc_error_codes::DIAGNOSTICS),  
    input: config::Input::File(path),  
}
```

## Just Use Rustc: More Config

```
lint_caps: rustc_hash::FxHashMap::default(),  
crate_cfg: rustc_hash::FxHashSet::default(),  
crate_check_cfg: config::CheckCfg::default(),  
input_path: None,  
output_dir: None,  
output_file: None,  
file_loader: None,  
parse_sess_created: None,  
register_lints: None,  
override_queries: None,  
make_codegen_backend: None,  
};
```

## Just Use Rustc: Session Globals and Tcx

```
// rustc_interface::interface::Compiler  
rustc_interface::run_compiler(config, |compiler| {  
    // rustc_interface::queries::Queries  
    compiler.enter(|queries| {  
        // rustc_middle::ty::context::TyCtxt  
        queries.global_ctxt().unwrap().take().enter(|tcx| {
```

## Just Use Rustc: Query HIR

```
let demo_fn_sym = Symbol::intern("demo_fn");

let hir_crate = tcx.hir();
for id in hir_crate.items() {
    let item = hir_crate.item(id);
    if let rustc_hir::ItemKind::Fn(sig, _gen, _body_id) = &item.kind {
        if item.ident.name == demo_fn_sym {
            println!("{sig:?}");
        }
    }
}
```

## Just Use Rustc: Input

```
pub fn demo_fn(x: i32, y: i32) -> i32 {  
    x + y  
}
```

## Just Use Rustc: Result

```
FnSig { header: FnHeader { unsafety: Normal, constness: NotConst, asyncness: Async, abi: Rust }, decl: FnDecl { inputs: [Ty { hir_id: HirId { owner: OwnerId { def_id: DefId(0:3 ~ demo[52ed]::demo_fn) }, local_id: 12 }, kind: Path(Resolved(None, Path { span: demo.rs:1:15: 1:18 (#0), res: PrimTy(Int(I32)), segments: [PathSegment { ident: i32#0, hir_id: HirId { owner: OwnerId { def_id: DefId(0:3 ~ demo[52ed]::demo_fn) }, local_id: 13 }, res: PrimTy(Int(I32)), args: None, infer_args: false }] })), span: demo.rs:1:15: 1:18 (#0) }, Ty { hir_id: HirId { owner: OwnerId { def_id: DefId(0:3 ~ demo[52ed]::demo_fn) }, local_id: 14 }, kind: Path(Resolved(None, Path { span: demo.rs:1:23: 1:26 (#0), res: PrimTy(Int(I32)), segments: [PathSegment { ident: i32#0, hir_id: HirId { owner: OwnerId { def_id: DefId(0:3 ~ demo[52ed]::demo_fn) }, local_id: 15 }, res: PrimTy(Int(I32)), args: None, infer_args: false }] })), span: demo.rs:1:23: 1:26 (#0) }], output: Ty { hir_id: HirId { owner: OwnerId { def_id: DefId(0:3 ~ demo[52ed]::demo_fn) }, local_id: 16 }, kind: Path(Resolved(None, Path { span: demo.rs:1:31: 1:34 (#0), res: PrimTy(Int(I32)), segments: [PathSegment { ident: i32#0, hir_id: HirId { owner: OwnerId { def_id: DefId(0:3 ~ demo[52ed]::demo_fn) }, local_id: 17 }, res: PrimTy(Int(I32)), args: None, infer_args: false }] })), span: demo.rs:1:31: 1:34 (#0) }
```



## Just Use Rustc: Result

```
FnSig {  
  header: FnHeader {  
    unsafety: Normal,  
    constness: NotConst,  
    asyncness: NotAsync,  
    abi: Rust,  
  },  
  inputs: [  
    Ty { kind: PrimTy(Int(I32)) },  
    Ty { kind: PrimTy(Int(I32)) },  
  ],  
  output: Return( Ty { kind: PrimTy(Int(I32)) } ),  
  c_variadic: false,  
  implicit_self: None,  
};
```

# Just Use Rustc Rustdoc



In demo

Functions

[demo\\_fn](#)

Click or press 'S' to search, '?' for more options...

?



Function **demo::demo\_fn** 

[source](#) · [-]

```
pub fn demo_fn(x: i32, y: i32) -> i32
```

## Just Use Rustc Rustdoc

```
<pre class="rust fn">
  <code>
    pub fn demo_fn(x:
      <a
        class="primitive"
        href="https://doc.rust-lang.org/nightly/std/primitive.i32.html"
      >i32</a>, y:
      <a
        class="primitive"
        href="https://doc.rust-lang.org/nightly/std/primitive.i32.html"
      >i32</a>) -&gt;
      <a
        class="primitive"
        href="https://doc.rust-lang.org/nightly/std/primitive.i32.html"
      >i32</a>
    </code>
  </pre>
```

## Just Use Rustc Rustdoc Rustdoc JSON

```
rustdoc +nightly demo.rs -Z unstable-options --output-format json  
oj -p 100.3 -x "$.index[?(@.name=='demo_fn')].inner" ./doc/demo.json
```

```
{  
  "decl": {  
    "c_variadic": false,  
    "inputs": [  
      ["x", {"inner": "i32", "kind": "primitive"}],  
      ["y", {"inner": "i32", "kind": "primitive"}]  
    ],  
    "output": {"inner": "i32", "kind": "primitive"}  
  },  
  "generics": {"params": [], "where_predicates": []},  
  "header": {"abi": "Rust", "async": false, "const": false, "unsafe": false}  
}
```

## Format: Item

```
pub struct Item {  
    pub id: Id,  
    pub name: Option<String>,  
    pub docs: Option<String>,  
    pub inner: ItemEnum,  
}
```

(All of these are simplified for clarity)

## Format: ItemEnum

```
pub enum ItemEnum {  
    Module(Module),  
    Import(Import),  
    Struct(Struct),  
    StructField(Type),  
    Enum(Enum),  
    Variant(Variant),  
    Function(Function),  
    Trait(Trait),  
    Method(Method),  
    Impl(Impl),  
    Typedef(Typedef),  
    Constant(Constant),  
}
```

## Format: Function / Method

```
pub struct Function {  
    pub decl: FnDecl,  
    pub generics: Generics,  
    pub header: Header,  
}  
  
pub struct Method {  
    pub decl: FnDecl,  
    pub generics: Generics,  
    pub header: Header,  
    pub has_body: bool,  
}  
  
pub struct Header {  
    pub const_: bool,  
    pub unsafe_: bool,  
    pub async_: bool,  
    pub abi: Abi,  
}
```

## Format: FnDecl

```
pub struct FnDecl {  
    pub inputs: Vec<(String, Type)>,  
    pub output: Option<Type>,  
    pub c_variadic: bool,  
}
```



## Format: Type

```
pub enum Type {  
    ResolvedPath(Path),  
    DynTrait(DynTrait),  
    Generic(String),  
    Primitive(String),  
    FunctionPointer(Box<FunctionPointer>),  
    Tuple(Vec<Type>),  
    Slice(Box<Type>),  
    Array { ... },  
    ImplTrait(Vec<GenericBound>),  
    Infer,  
    RawPointer { ... },  
    BorrowedRef { ... },  
    QualifiedPath { ... },  
}
```

## Format: Type Continued

```
Array {  
    type_: Box<Type>,  
    len: String,  
},  
RawPointer {  
    mutable: bool,  
    type_: Box<Type>,  
},  
QualifiedPath {  
    name: String,  
    args: Box<GenericArgs>,  
    self_type: Box<Type>,  
    trait_: Path,  
}
```

These should probably be structs.

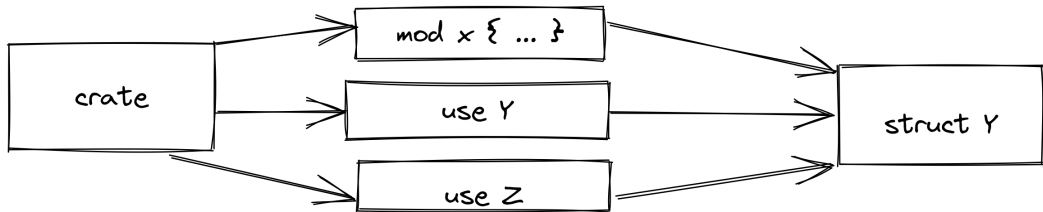
## Format: Module / Crate

```
pub struct Module {  
    pub items: Vec<Id>,  
}  
  
pub struct Id(pub String);  
  
pub struct Crate {  
    pub root: Id,  
    pub index: HashMap<Id, Item>,  
    pub format_version: u32,  
}
```

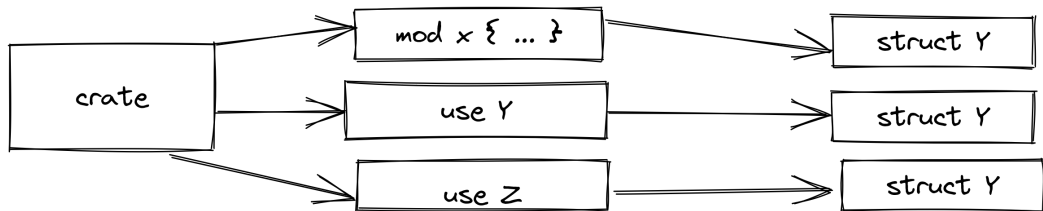
## Why the IDs? Graphs

```
pub mod x {  
    pub struct Y;  
}  
pub use x::Y;  
pub use x::Y as Z;
```

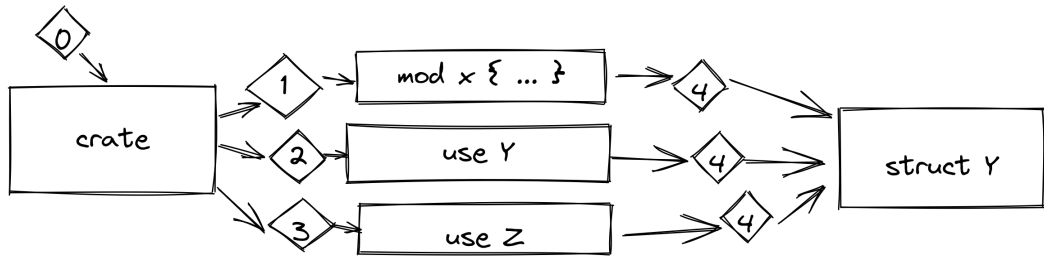
## Why the IDs? Graphs



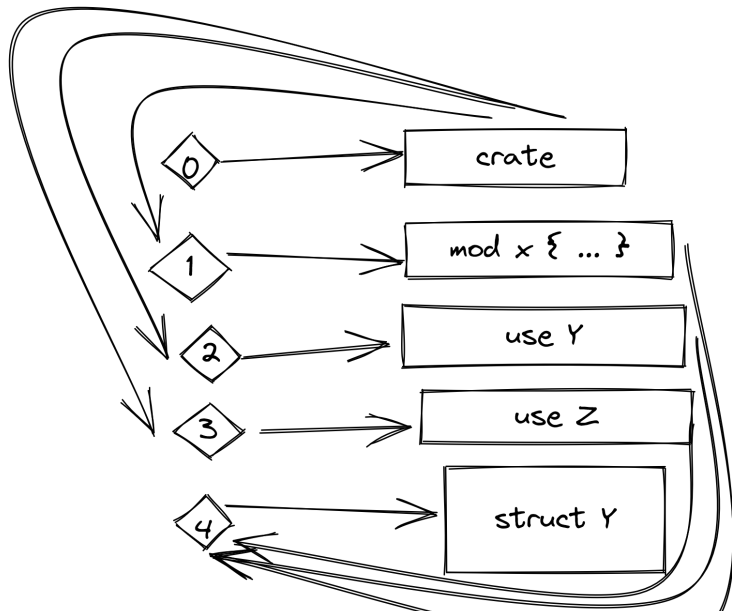
## Why the IDs? Graphs



## Why the IDs? Graphs

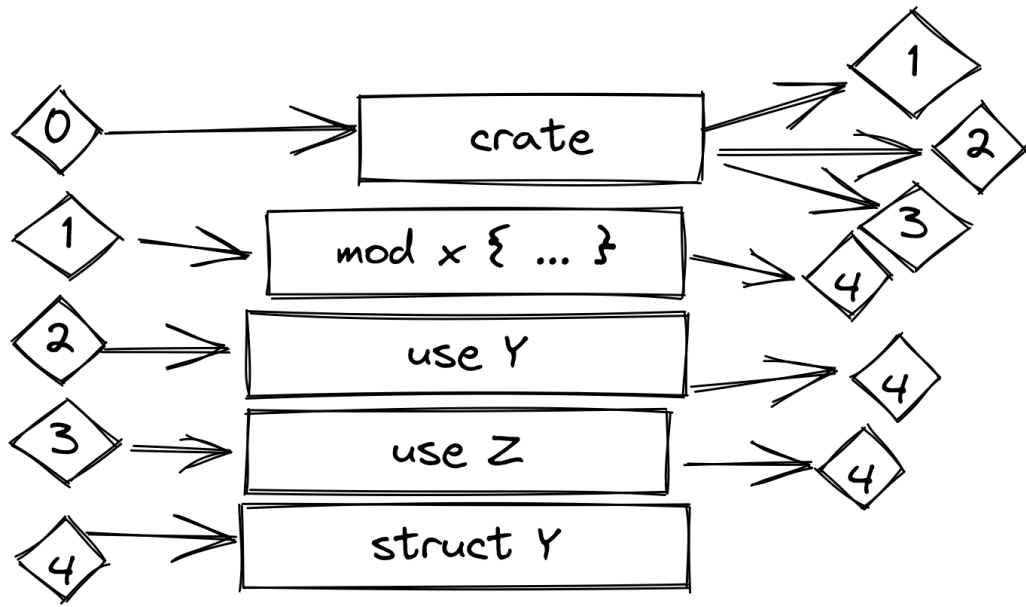


## Why the IDs? Graphs





## Why the IDs? Graphs



## Why the IDs? Graphs

```
pub mod x {  
    pub struct Y;  
}  
pub use x::Y;  
pub use x::Y as Z;
```

## JSON Output

```
{
  "index": {
    "0": {
      "inner": {"items": ["1", "2", "3"]},
      "kind": "module",
      "name": "cratename"
    },
    "1": {"inner": {"items": ["4"]}, "kind": "module", "name": "x"},
    "2": {"inner": {"id": "4", "name": "Y"}, "kind": "import", "name": null},
    "3": {"inner": {"id": "4", "name": "Z"}, "kind": "import", "name": null},
    "4": {"inner": {}, "kind": "struct", "name": "Y"}
  },
  "root": "0"
}
```

# Questions

## Thanks

Alex Kladoy, Didrik Nordström, Guillaume Gomez, Jacob Hoffman-Andrews, Joseph Ryan, Joshua Nelson, León Orell Valerian Liehr, Luca Palmieri, Martin Nordholts, Michael Goulet, Michael Howell, Noah Lev, QuietMisdreavus, Rune Tynan, Tyler Mandry, Urgau  
Excalidraw icons by xxxDeveloper.

## Code

- ▶ `src/librustdoc/json`
- ▶ `src/rustdoc-json-types`
- ▶ `src/tools/jsdocck`
- ▶ `src/tools/jsdoclint`
- ▶ `docs.rs/rustdoc-types/`

## Contact

- ▶ `nixon.emoony@gmail.com`