

## Design and Implementation of the #[diagnostic] namespace RustLab 2024

## Introduction



- Developer and Researcher at GiGa infosystems
- Write Rust for 10 years
- Maintainer of Diesel







- Rustc is known for good error messages
- Nevertheless can produce large hard to understand errors.
- Rustc relies on heuristics to decide what's relevant to show
- Often crate authors do know what's relevant and what's not

 $\implies$  Providing a way to customize error messages would allow crate authors to significantly improve the developer experience for their crates

# The #[diagnostic] namespace



 $\blacktriangleright$  Tool attribute namespace  $\longrightarrow$  modifies behaviour of a rust tool

Contains a set of attributes to modify error messages emitted by the compiler

#[diagnostic::on\_unimplemented(message = "MyCustomError")]
trait Foo {}

#[diagnostic::do\_not\_recommend]
impl<T> Foo for T where T: Send {}

# The #[diagnostic] namespace: Rules



- Any attribute in there is a suggestion to the compiler
- Unknown attributes are ignored
- Malformed attributes are ignored
- Compiler might change the output later, no guarantees that the hint is applied
- Attributes are not allowed to change the compilation result
- New attributes can be easily added without requiring a full RFC



- Stable since 1.78 (May 2024)
- Modifies error message emitted if a trait bound is not satisfied
- Allows to overwrite multiple parts of the error message: message, label and notes
- Support using format place holders for generics



```
trait ImportantTrait<A> {}
```

}

```
fn use_my_trait(_: impl ImportantTrait<i32>) {}
```

```
fn main() {
    use_my_trait(String::new());
```



```
error[E0277]: the trait bound `String: ImportantTrait<i32>` is
    not satisfied
 --> src/main.rs:6:18
6 I
        use my trait(String::new());
                                   the trait `ImportantTrait<i32>` is
                                   not implemented for `String`
        required by a bound introduced by this call
help: this trait has no implementations, consider adding one
 --> src/main.rs:1:1
    trait ImportantTrait<A> {}
```

q

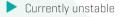
```
#[diagnostic::on unimplemented(
    message = "My Message for `ImportantTrait<{A}>` not \
               implemented for `{Self}`",
    label = "My Label".
    note = "Note 1",
    note = "Note 2"
)1
trait ImportantTrait<A> {}
fn use my trait( : impl ImportantTrait<i32>) {}
fn main() {
   use my trait(String::new());
7
```





```
error[E0277]: My Message for `ImportantTrait<i32>`
     not implemented for `String`
  --> src/main.rs:14:18
14 I
        use my trait(String::new());
                                    My Label
         required by a bound introduced by this call
   = help: the trait `ImportantTrait<i32>` is not implemented for `String`
   = note: Note 1
   = note: Note 2
```



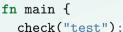


Proposed stabilisation for 1.84 (January 2025)

Hides certain trait implementations from error messages

```
trait Expression {
   type SqlType;
}
trait AsExpression<ST> {}
```

```
impl<T, ST> AsExpression<ST> for T
where T: Expression<SqlType = ST>
{ /* ... */ }
impl AsExpression<Integer> for i32 { /* ... */ }
impl AsExpression<Text> for String { /* ... */ }
fn check(_: impl AsExpression<Integer>) {}
```



}





```
error[E0277]: the trait bound `&str: Expression` is not satisfied
    --> src/main.rs:53:15
```

```
LL | check("test");

| ^^^^ the trait `Expression` is not implemented for

| `&str`, which is required by `&str: AsExpression<Integer>`

note: required for `&str` to implement `AsExpression<Integer>`

--> src/main.rs:26:13
```

LL | impl<T, ST> AsExpression<ST> for T

```
LL | where
```

LL | T: Expression<SqlType = ST>,

----- unsatisfied trait bound introduced here



#### **Combinations**



```
error[E0277]: the trait bound `posts::columns::id: SelectableExpression<users::table>` is not satisfied
  --> tests/diesel/invalid guery.rs:20:18
20
        users::table.select(posts::id);
                      ^^^^^ the trait 'SelectableExpression<users::table>' is not implemented for
`posts::columns::id`
  = help: the following other types implement trait 'SelectableExpression<OS>':
             <posts::columns::id as SelectableExpression<JoinOn<Join. On>>>
             <posts::columns::id as SelectableExpression<Onlv<posts::table>>>
             consts::columns::id as SelectableExpression<SelectStatement<FromClause<From>>>>
             cposts::columns::id as SelectableExpression(diesel::internal::table macro::Join(Left, Right,
Inner>>>
             <costs::columns::id as SelectableExpression<diesel::internal::table macro::Join<Left. Right.</pre>
LeftOuter>>>
             cposts::columns::id as SelectableExpression<posts::table>>
  = note: required because of the requirements on the impl of `SelectDsl<posts::columns::id>` for
'SelectStatement<FromClause<users::table>>'
```

Figure 1: Old error

```
20 | users::table.select(posts::id);
```

```
= note: `posts::columns::id` is no valid selection for `users::table`
```

= note: required because of the requirements on the impl of `SelectDsl<posts::columns::id>` for `SelectStatement<FromClause<users::table>>`

#### Figure 2: New error

## Implementation



- **1.** Identify a problem
- 2. Write an RFC to propose a solution
- **3.** Work on the implementation
- **4.** Stabilize the feature

## The problem: Complicated error messages



- Certain crates rely on the type system to model invariants
- Rustc emits a generic error message if that invariant is violated
- Crate authors can often provide specific information what is wrong there and provide pointers how to resolve the problem
- Putting these hints and suggestions in the documentation makes it harder for users to discover
  - Putting this information in the error message displays them right in front of the user

## Writing an RFC





#### Needs to include:

- Motivation
- Design of the new feature
- Drawbacks
- Possible alternatives
- Unresolved questions
- Not every RFC is accepted by the relevant team
- An accepted RFC does not automatically give you a stable feature

The #[diagnostic] attribute namespace #3368	
- Merged oli-obk merged 10 commits into rust-lang:master from weiznich:diagnostic_attribute_namespace	🧧 🔁 on May 26, 2023
Conversation 88	
weiznich commented on Jan 6, 2023	Contributor ····
Summary	
The RC properts to add a common statements; and the namespace for attributes that can influence the error message entited by the complex it specifies a set of rules what statisticates in this namespace are allowed to do how these attributes must be handled by the complex and anise it disallowed in this namespace. In addition this RC properson a statement is inclusive tensories, attribute to influence error messages entitled by unstatified traits bounds.	
I would like to use this possibility to thank the rust foundation for supporting my work on this RFC throu-	gh a project grant.
Rendered	
▲ 2 ♥ 1 ♥ 19 ₩ 7	
-0- 🖲 The #[dispestic] attribute namespace	Verified b9bf6ad

## Implementing the new language/compiler feature



- 1. Introduce a new nightly feature
- 2. Implement the new feature in the compiler
- 3. Write tests for the new feature
- 4. Start using the feature in the ecosystem and possibly in the Rust standard library

#### Implementation strategies



- The Rust compiler code base is just a large Rust code base
- You don't need to understand everything, just that little part you are working on
- $\blacktriangleright$  Rustc has a good testing setup for diagnostics ightarrow Easy to add test cases first to inspect the result
  - Searching for error messages gives you an entry point for where to start looking
  - println! based debug strategies work well to understand what's going on

## **Stabilization**



- Summarize the changes made during implementation
- Add documentation of the new feature to the Rust Reference
- Remove the unstable feature usage
- Wait for the relevant team to take a decision
- Not every item proposed for stabilization will be stabilized

#### Stabilize the #[diagnostic] namespace and #[diagnostic::on\_unimplemented] attribute #119888

⊱ Merged bors merged 1 commit into rust-lang:master from woiznich:stablize\_diagnostic\_namespace 🗗 on Mar 8

Conversation 31 -o- Commits 1 P. Checks 11 E Files changed 27

weiznich commented on Jan 12 • edited by compiler-errors +

Contributor ...

This PR stabilizes the a[diagnostic] attribute namespace and a minimal option of the a[diagnostic::on\_unimplemented] attribute.

The (it(is)) attribute namespace is meant to provide a home for attributes that allow users to influence error messages emitted by the compiler. The compiler is not guaranteed to use any of this hirts, however it should accept any (non-positing attribute in this namespace and potentially emit line warnings for unused attributes and options. This is meant to allow discarding certain attributes/options in the future to allow fundamental changes to the compiler without the need to keep then non-manifold options working.

The #(diagnostic::on\_unisplemented) attribute is allowed to appear on a trait definition. This allows crate authors to hint the compiler to emit a specific error message if a certain trait is not implemented. For the #(diagnostic:me.unsplemented) attribute the following options are implemented:

- · message which provides the text for the top level error message
- · Tabe1 which provides the text for the label shown inline in the broken code in the error message
- · note which provides additional notes.

The note option can appear several times, which results in several note messages being emitted. If any of the other options appears several times the first occurrence of the relevant option specifies the actually used value. Any other occurrence generates an init warming. For any other non-existing option a init warming is generated.

All three options accept a text as argument. This text is allowed to contain format parameters referring to generic argument or soil: by name via the (soil) or (biaeoffenericArgument) syntax. For any non-existing argument a lint warring is generated.

This allows to have a trait definition like:

```
#iddport(internet) internet()
internet) internet()
internet) internet()
internet)
```



- The #[diagnostic] namespace is home for attributes providing hints to influence compiler error messages
- The #[diagnostic::on\_unimplemented] attribute allows to change the error message emitted for unimplemented traits
- The #[diagnostic::do\_not\_recommend] attribute allows to hide trait implementations from error messages
- Overall helps to make compiler error messages in Rust even better
- Adding new attributes does not require a RFC, but just an implementation