



Leigh McCulloch
Principal Software Engineer
Stellar Development Foundation

Inside Stellar

The Rust Behind Stellar's Smart Contracts

soroban-sdk

```
#[contract]
struct Token;

#[contractimpl]
impl Token {
    pub fn transfer(
        from: Address,
        to: Address,
        amount: i128,
    ) { ... }
}
```

Macros generate code.

```
#[contract]
struct Token;

#[contractimpl]
impl Token {
    pub fn transfer(
        from: Address,
        to: Address,
        amount: i128,
    ) { ... }
}
```

```
impl Token {
    pub fn transfer(...) { ... }
}

#[export_name = "transfer"]
pub extern "C" fn __Token__transfer__(
    arg_0: soroban_sdk::Val,
    arg_1: soroban_sdk::Val,
    arg_2: soroban_sdk::Val,
) → soroban_sdk::Val {
    Token::transfer(...)
}
```

This impl block is empty. How does the Wasm export three functions?

crate **token** (our contract)

```
use pause::Pause;

#[contractimpl(contracttrait)]
impl Pause for Token {}
```

stellar contract info interface

```
fn paused() → bool
```

```
fn pause()
```

```
fn unpause()
```

This impl block is empty. How does the Wasm export three functions?

crate **token** (our contract)

```
use pause::Pause;

#[contractimpl(contracttrait)]
impl Pause for Token {}
```

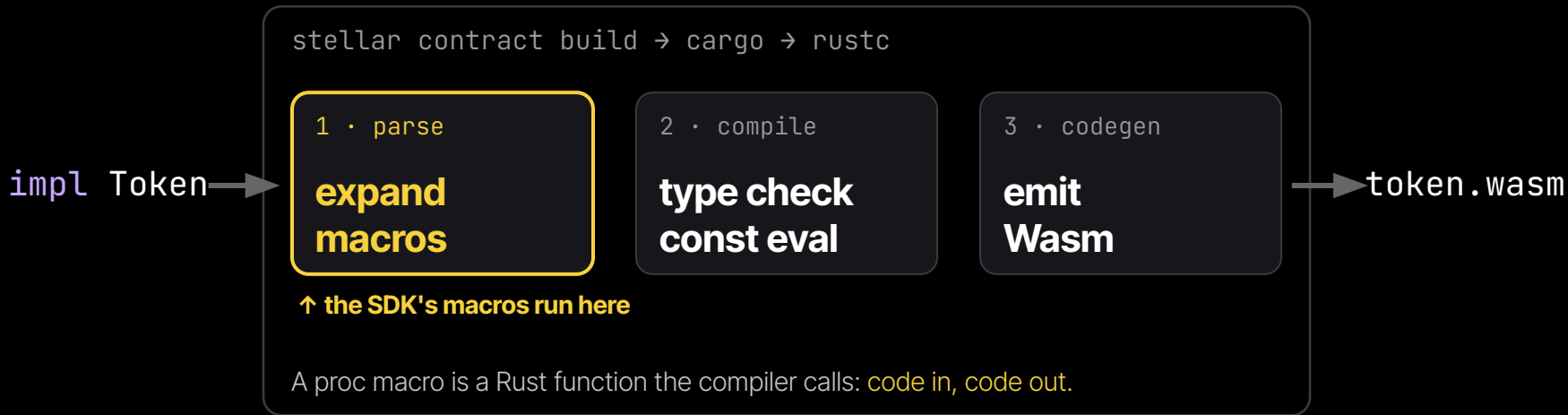
stellar contract info interface

```
fn paused() → bool
```

```
fn pause()
```

```
fn unpause()
```

Macros run inside the build.



Proc macros are powerful, but they've got limitations.

01 · visibility

Macros only see their own items

→ recursive expansion

02 · types

Macros see code, not types

→ const evaluation

03 · coordination

Macros can't combine results across calls

→ module init functions

#[contractimpl] can't see the trait.

crate **pause**

```
#[contracttrait]
pub trait Pause {
    fn paused() → bool { ... }
    fn pause() { ... }
    fn unpause() { ... }
}
```

crate **token** (our contract)

```
use pause::Pause;

#[contractimpl(contracttrait)]
impl Pause for Token {}
```

proc → declarative → proc

crate **pause**

```
#[contracttrait]
pub trait Pause { ... }
```



generates, with the defaults baked in

```
macro_rules! Pause {
    paused, pause, unpause
}
```

crate **token** (our contract)

```
#[contractimpl(contracttrait)]
impl Pause for Token {}
```

proc → declarative → proc

crate **pause**

```
#[contracttrait]
pub trait Pause { ... }
```

↓ generates, with the defaults baked in

```
macro_rules! Pause {
  paused, pause, unpause
}
```

crate **token** (our contract)

```
#[contractimpl(contracttrait)]
impl Pause for Token {}
```

↓ calls, with overridden fns

```
Pause!(Token, overridden: [])
```

↓ passes both lists to

```
contractimpl_trait_default_fns_
not_overridden!
```

↓ generates exports, spec, client

```
paused()
```

```
pause()
```

```
unpause()
```

proc → declarative → proc

crate **pause**

```
#[contracttrait]
pub trait Pause { ... }
```



generates, with the defaults baked in

```
macro_rules! Pause {
    paused, pause, unpause
}
```



crate **token** (our contract)

```
#[contractimpl(contracttrait)]
impl Pause for Token {}
```



calls, with overridden fns

```
Pause!(Token, overridden: [])
```



passes both lists to

```
contractimpl_trait_default_fns_
not_overridden!
```



generates exports, spec, client

```
paused()
```

```
pause()
```

```
unpause()
```

proc → declarative → proc

crate **pause**

```
#[contracttrait]
pub trait Pause { ... }
```

↓ generates, with the defaults baked in

```
macro_rules! Pause {
    paused, pause, unpause
}
```

crate **token** (our contract)

```
#[contractimpl(contracttrait)]
impl Pause for Token {}
```

↓ calls, with overridden fns

```
Pause!(Token, overridden: [])
```

↓ passes both lists to

```
contractimpl_trait_default_fns_
not_overridden!
```

↓ generates exports, spec, client

`paused()`

`pause()`

`unpause()`

Macros see code, not types.

crate **token** (our contract)

```
use soroban_sdk::Address;  
type Destination = Address;  
  
#[contractevent]  
pub struct Transfer {  
    #[topic] from: Address,  
    #[topic] to: Destination,  
    amount: i128,  
}
```

Macro sees

"Transfer"

"Address"

"Destination"

Compiler knows

token::Transfer

soroban_sdk::Address

soroban_sdk::Address

Macros see code, not types.

crate **token** (our contract)

```
use soroban_sdk::Address;
type Destination = Address;

#[contractevent]
pub struct Transfer {
    #[topic] from: Address,
    #[topic] to: Destination,
    amount: i128,
}
```

Macro sees

"Transfer"

"Address"

"Destination"

Compiler knows

token::Transfer

soroban_sdk::Address

soroban_sdk::Address



Macros see code, not types.

crate **token** (our contract)

```
use soroban_sdk::Address;
type Destination = Address;

#[contractevent]
pub struct Transfer {
    #[topic] from: Address,
    #[topic] to: Destination,
    amount: i128,
}
```

Macro sees

"Transfer"

"Address"

"Destination"

Compiler knows

token::Transfer

soroban_sdk::Address

soroban_sdk::Address

Macros plus const expressions.

```
const ENTRY: ScSpecEntry = ScSpecEntry::EventV0(ScSpecEventV0 {
    name: Transfer::SPEC_NAME, // "token::Transfer"
    params: [
        { name: "from", type: Address::SPEC_TYPE_DEF, ... },
        { name: "to",   type: Destination::SPEC_TYPE_DEF, ... }, ...
    ], ...
});
```

```
#[link_section = "contractspev0"]
static ENTRY_XDR: [u8; ENTRY.const_xdr_len()] = ENTRY.const_to_xdr();
```

```
impl SpecName for Transfer {
    const SPEC_NAME: &str = concat!(module_path!(), "::Transfer");
}
```

Macros plus const expressions.

```
const ENTRY: ScSpecEntry = ScSpecEntry::EventV0(ScSpecEventV0 {
    name: Transfer::SPEC_NAME, // "token::Transfer"
    params: [
        { name: "from", type: Address::SPEC_TYPE_DEF, ... },
        { name: "to",   type: Destination::SPEC_TYPE_DEF, ... }, ...
    ], ...
});
```

```
#[link_section = "contractspecv0"]
static ENTRY_XDR: [u8; ENTRY.const_xdr_len()] = ENTRY.const_to_xdr();
```

```
impl SpecTypeDef for Address {
    const SPEC_TYPE_DEF: ScSpecTypeDef = ScSpecTypeDef::Address;
}
```



Macros plus const expressions.

```
const ENTRY: ScSpecEntry = ScSpecEntry::EventV0(ScSpecEventV0 {
    name: Transfer::SPEC_NAME, // "token::Transfer"
    params: [
        { name: "from", type: Address::SPEC_TYPE_DEF, ... },
        { name: "to",   type: Destination::SPEC_TYPE_DEF, ... }, ...
    ], ...
});

#[link_section = "contractspev0"]
static ENTRY_XDR: [u8; ENTRY.const_xdr_len()] = ENTRY.const_to_xdr();
```

```
impl SpecTypeDef for Address {
    const SPEC_TYPE_DEF: ScSpecTypeDef = ScSpecTypeDef::Address;
}
```

Each macro sees one block. None sees them all.

```
#[contractimpl]
impl Token {
  pub fn transfer(...) { ... }
}
```

transfer

```
#[contractimpl(contracttrait)]
impl Pause for Token {}
```

paused
pause,
unpause

Native tests need a name → fn map.
No macro sees all fns.

```
#[test]
fn test_transfer() {
  let env = Env::default();
  let addr = env.register(Token, ());
  let client = TokenClient::new(&env, &addr);
  ...
  // Call transfer function:
  client.transfer(&from, &to, &10);
}
```

Each impl registers its functions before main, before tests.

```
// From impl Token
#[ctor] fn ...() {
    FUNCS.insert("transfer", ...);
}
```

```
// From impl Pause for Token
#[ctor] fn ...() {
    FUNCS.insert("paused", ...);
    FUNCS.insert("pause", ...);
    FUNCS.insert("unpause", ...);
}
```

```
"transfer" → fn
"paused"   → fn
"pause"    → fn
"unpause"  → fn
```

```
#[test]
fn test_transfer() {
    ...
    client.transfer(&from, &to, &10);
}
```

Each impl registers its functions before main, before tests.

```
// From impl Token
#[ctor] fn ...() {
    FUNCS.insert("transfer", ...);
}
```

```
// From impl Pause for Token
#[ctor] fn ...() {
    FUNCS.insert("paused", ...);
    FUNCS.insert("pause", ...);
    FUNCS.insert("unpause", ...);
}
```

```
"transfer" → fn
"paused"   → fn
"pause"    → fn
"unpause"  → fn
```

```
#[test]
fn test_transfer() {
    ...
    client.transfer(&from, &to, &10);
}
```

Each impl registers its functions before main, before tests.

```
// From impl Token
#[ctor] fn ...() {
    FUNCS.insert("transfer", ...);
}
```

```
// From impl Pause for Token
#[ctor] fn ...() {
    FUNCS.insert("paused", ...); ...
}
```

```
"transfer" → fn
"paused"   → fn
"pause"    → fn
"unpause"  → fn
```

```
#[test]
fn test_transfer() {
    ...
    client.transfer(&from, &to, &10);
}
```

```
// In the soroban-env
invoke(addr, "transfer", args)
```

By using these Rust features with macros, the SDK enables

`recursive expansion`

**Code sharing
with traits
across crates
and libraries**

`const evaluation`

**A contract
interface that
has strong type
identity**

`module init functions`

**Fast,
step-through
debuggable
Rust tests**

Patterns for your own macros

recursive expansion

Need data from another item or crate?

Generate a macro as a data channel.

```
macro_rules! Pause {...}
```

const evaluation

Need to know what a type really is?

Emit a const, let the compiler evaluate.

```
const SPEC: _ = ...;
```

module init functions

Need to combine results from many macro calls?

Have each register a ctor.

```
#[ctor] fn reg() { ... }
```



Each has its own limits and traps. Read up before using them.

Check out the code

`github.com/stellar/rs-soroban-sdk`

`soroban-sdk-macros/`

`tests-expanded/`

`soroban-sdk/docs/contracttrait.md`