The Move Book

Witness is a pattern of proving an existence by constructing a proof. In the context of programming, witness is a way to prove a certain property of a system by providing a value that can only be constructed if the property holds.

In the Struct section we have shown that a struct can only be created - or packed - by the module defining it. Hence, in Move, a module proves ownership of the type by constructing it. This is one of the most important patterns in Move, and it is widely used for generic type instantiation and authorization.

Practically speaking, for the witness to be used, there has to be a function that expects a witness as an argument. In the example below it is the new function that expects a witness of the T type to create a Instance instance.

It is often the case that the witness struct is not stored, and for that the function may require the Drop ability for the type.

The only way to construct an Instance is to call the new function with an instance of the type T . This is a basic example of the witness pattern in Move. A module providing a witness often has a matching implementation, like the module book::witness_source below:

The instance of the struct W is passed into the new_instance function to create an Instance , thereby proving that the module book::witness_source owns the type W .

Witness allows generic types to be instantiated with a concrete type. This is useful for inheriting associated behaviors from the type with an option to extend them, if the module provides the ability to do so.

In the example above, which is borrowed from the balance module of the Sui Framework , the Supply a generic struct that can be constructed only by supplying a witness of the type T . The witness is taken by value and discarded - hence the T must have the drop ability.

The instantiated Supply can then be used to mint new Balance 's, where T is the type of the supply.

While a struct can be created any number of times, there are cases where a struct should be guaranteed to be created only once. For this purpose, Sui provides the "One-Time Witness" - a special witness that can only be used once. We explain it in more detail in the next section .

In the next section, we will learn about the One Time Witness pattern.

Witness in Move

In the Struct section we have shown that a struct can only be created - or packed - by the module defining it. Hence, in Move, a module proves ownership of the type by constructing it. This is one of the most important patterns in Move, and it is widely used for generic type instantiation and authorization.

Practically speaking, for the witness to be used, there has to be a function that expects a witness as an argument. In the example below it is the new function that expects a witness of the T type to create a Instance instance.

It is often the case that the witness struct is not stored, and for that the function may require the Drop ability for the type.

```bash module book::witness;

/// A struct that requires a witness to be created. public struct Instance { t: T }

/// Create a new instance of `Instance<T>` with the provided T. public fun new(witness: T): Instance { Instance { t: witness } } ```

The only way to construct an Instance is to call the new function with an instance of the type T . This is a basic example of the witness pattern in Move. A module providing a witness often has a matching implementation, like the module book::witness_source below:

```bash module book::witness_source;

use book::witness::{Self, Instance};

/// A struct used as a witness. public struct W {}

/// Create a new instance of `Instance<W>`. public fun new_instance(): Instance { witness::new(W {}) } ```

The instance of the struct W is passed into the new_instance function to create an Instance , thereby proving that the module book::witness_source owns the type W .

Witness allows generic types to be instantiated with a concrete type. This is useful for inheriting associated behaviors from the type with an option to extend them, if the module provides the ability to do so.

```bash // File: sui-framework/sources/balance.move /// A Supply of T. Used for minting and burning. public struct Supply has store { value: u64, }

/// Create a new supply for type T with the provided witness. public fun create_supply(_w: T): Supply { Supply { value: 0 } }

/// Get the `Supply` value. public fun supply_value(supply: &Supply): u64 { supply.value } ```

In the example above, which is borrowed from the balance module of the Sui Framework , the Supply a generic struct that can be constructed only by supplying a witness of the type T . The witness is taken by value and discarded - hence the T must have the drop ability.

The instantiated Supply can then be used to mint new Balance 's, where T is the type of the supply.

```bash // File: sui-framework/sources/balance.move /// Storable balance. public struct Balance has store { value: u64, }

/// Increase supply by `value` and create a new `Balance<T>` with this value. public fun increase_supply(self: &mut Supply, value: u64): Balance { assert!(value < (18446744073709551615u64 - self.value), EOverflow); self.value = self.value + value; Balance { value } } ```

While a struct can be created any number of times, there are cases where a struct should be guaranteed to be created only once. For this purpose, Sui provides the "One-Time Witness" - a special witness that can only be used once. We explain it in more detail in the next section .

In the next section, we will learn about the One Time Witness pattern.

## Instantiating a Generic Type

Witness allows generic types to be instantiated with a concrete type. This is useful for inheriting associated behaviors from the type with an option to extend them, if the module provides the ability to do so.

```bash // File: sui-framework/sources/balance.move /// A Supply of T. Used for minting and burning. public struct Supply has store { value: u64, }

/// Create a new supply for type T with the provided witness. public fun create_supply(_w: T): Supply { Supply { value: 0 } }

/// Get the `Supply` value. public fun supply_value(supply: &Supply): u64 { supply.value } ```

In the example above, which is borrowed from the balance module of the Sui Framework , the Supply a generic struct that can be constructed only by supplying a witness of the type T . The witness is taken by value and discarded - hence the T must have the drop ability.

The instantiated Supply can then be used to mint new Balance 's, where T is the type of the supply.

```bash // File: sui-framework/sources/balance.move /// Storable balance. public struct Balance has store { value: u64, }

/// Increase supply by `value` and create a new `Balance<T>` with this value. public fun increase_supply(self: &mut Supply, value: u64): Balance { assert!(value < (18446744073709551615u64 - self.value), EOverflow); self.value = self.value + value; Balance { value } } ```

While a struct can be created any number of times, there are cases where a struct should be guaranteed to be created only once. For this purpose, Sui provides the "One-Time Witness" - a special witness that can only be used once. We explain it in more detail in the next section .

In the next section, we will learn about the One Time Witness pattern.

## One Time Witness

While a struct can be created any number of times, there are cases where a struct should be guaranteed to be created only once.

For this purpose, Sui provides the "One-Time Witness" - a special witness that can only be used once. We explain it in more detail in the next section .

In the next section, we will learn about the One Time Witness pattern.

## Summary

In the next section, we will learn about the One Time Witness pattern.

## Next Steps

In the next section, we will learn about the One Time Witness pattern.