The Move Book

While regular Witness is a great way to statically prove the ownership of a type, there are cases where we need to ensure that a Witness is instantiated only once. And this is the purpose of the One Time Witness (OTW).

The OTW is a special type of Witness that can be used only once. It cannot be manually created and it is guaranteed to be unique per module. Sui Adapter treats a type as an OTW if it follows these rules:

Here is an example of an OTW:

The OTW cannot be constructed manually, and any code attempting to do so will result in a compilation error. The OTW can be received as the first argument in the module initializer . And because the init function is called only once per module, the OTW is guaranteed to be instantiated only once.

To check if a type is an OTW, sui::types module of the Sui Framework offers a special function is_one_time_witness that can be used to check if the type is an OTW.

The OTW pattern is a great way to ensure that a type is used only once. Most of the developers should understand how to define and receive the OTW, while the OTW checks and enforcement is mostly needed in libraries and frameworks. For example, the sui::coin module requires an OTW in the coin::create_currency method, therefore enforcing that the coin::TreasuryCap is created only once.

OTW is a powerful tool that lays the foundation for the Publisher object, which we will cover in the next section.

Definition

The OTW is a special type of Witness that can be used only once. It cannot be manually created and it is guaranteed to be unique per module. Sui Adapter treats a type as an OTW if it follows these rules:

Here is an example of an OTW:

```bash module book::one_time;

/// The OTW for the `book::one_time` module. /// Only `drop`, no fields, no generics, all uppercase. public struct ONE_TIME has drop {}

/// Receive the instance of `ONE_TIME` as the first argument. fun init(otw: ONE_TIME, ctx: &mut TxContext) { // do something with the OTW } ```

The OTW cannot be constructed manually, and any code attempting to do so will result in a compilation error. The OTW can be received as the first argument in the module initializer . And because the init function is called only once per module, the OTW is guaranteed to be instantiated only once.

To check if a type is an OTW, sui::types module of the Sui Framework offers a special function is_one_time_witness that can be used to check if the type is an OTW.

```bash use sui::types;

const ENotOneTimeWitness: u64 = 1;

/// Takes an OTW as an argument, aborts if the type is not OTW. public fun takes_witness(otw: T) { assert! (types::is_one_time_witness(&otw), ENotOneTimeWitness); } ```

The OTW pattern is a great way to ensure that a type is used only once. Most of the developers should understand how to define and receive the OTW, while the OTW checks and enforcement is mostly needed in libraries and frameworks. For example, the sui::coin module requires an OTW in the coin::create_currency method, therefore enforcing that the coin::TreasuryCap is created only once.

OTW is a powerful tool that lays the foundation for the Publisher object, which we will cover in the next section.

## Enforcing the OTW

To check if a type is an OTW, sui::types module of the Sui Framework offers a special function is_one_time_witness that can be

used to check if the type is an OTW.

```bash use sui::types;

const ENotOneTimeWitness: u64 = 1;

/// Takes an OTW as an argument, aborts if the type is not OTW. public fun takes_witness(otw: T) { assert! (types::is_one_time_witness(&otw), ENotOneTimeWitness); } ```

The OTW pattern is a great way to ensure that a type is used only once. Most of the developers should understand how to define and receive the OTW, while the OTW checks and enforcement is mostly needed in libraries and frameworks. For example, the sui::coin module requires an OTW in the coin::create_currency method, therefore enforcing that the coin::TreasuryCap is created only once.

OTW is a powerful tool that lays the foundation for the Publisher object, which we will cover in the next section.

## Summary

The OTW pattern is a great way to ensure that a type is used only once. Most of the developers should understand how to define and receive the OTW, while the OTW checks and enforcement is mostly needed in libraries and frameworks. For example, the sui::coin module requires an OTW in the coin::create_currency method, therefore enforcing that the coin::TreasuryCap is created only once.

OTW is a powerful tool that lays the foundation for the Publisher object, which we will cover in the next section.
```