The Move Book

A common use case in many applications is to run certain code just once when the package is published. Imagine a simple store module that needs to create the main Store object upon its publication. In Sui, this is achieved by defining an init function within the module. This function will automatically be called when the module is published.

All of the modules' init functions are called during the publishing process. Currently, this behavior is limited to the publish command and does not extend to package upgrades.

In the same package, another module can have its own init function, encapsulating distinct logic.

The function is called on publish, if it is present in the module and follows the rules:

TxContext can also be passed as immutable reference: &TxContext . However, practically speaking, it should always be &mut TxContext since the init function can't access the on-chain state and to create new objects it requires the mutable reference to the context.

While init function can be used to create sensitive objects once, it is important to know that the same object (eg. StoreOwnerCap from the first example) can still be created in another function. Especially given that new functions can be added to the module during an upgrade. So the init function is a good place to set up the initial state of the module, but it is not a security measure on its own.

There are ways to guarantee that the object was created only once, such as the One Time Witness . And there are ways to limit or disable the upgrade of the module, which we will cover in the Package Upgrades chapter.

As follows from the definition, the init function is guaranteed to be called only once when the module is published. So it is a good place to put the code that initializes module's objects and sets up the environment and configuration.

For example, if there's a Capability which is required for certain actions, it should be created in the init function. In the next chapter we will talk about the Capability pattern in more detail.

init

The function is called on publish, if it is present in the module and follows the rules:

```
bash fun init(ctx: &mut TxContext) { /* ... */} fun init(otw: OTW, ctx: &mut TxContext) { /* ... */
}
```

TxContext can also be passed as immutable reference: &TxContext . However, practically speaking, it should always be &mut TxContext since the init function can't access the on-chain state and to create new objects it requires the mutable reference to the context.

```
bash fun init(ctx: &TxContext) { /* ... */} fun init(otw: OTW, ctx: &TxContext) { /* ... */ }
```

While init function can be used to create sensitive objects once, it is important to know that the same object (eg. StoreOwnerCap from the first example) can still be created in another function. Especially given that new functions can be added to the module during an upgrade. So the init function is a good place to set up the initial state of the module, but it is not a security measure on its own.

There are ways to guarantee that the object was created only once, such as the One Time Witness . And there are ways to limit or disable the upgrade of the module, which we will cover in the Package Upgrades chapter.

As follows from the definition, the init function is guaranteed to be called only once when the module is published. So it is a good place to put the code that initializes module's objects and sets up the environment and configuration.

For example, if there's a Capability which is required for certain actions, it should be created in the init function. In the next chapter we will talk about the Capability pattern in more detail.

Trust and security

While init function can be used to create sensitive objects once, it is important to know that the same object (eg. StoreOwnerCap from the first example) can still be created in another function. Especially given that new functions can be added to the module during an upgrade. So the init function is a good place to set up the initial state of the module, but it is not a security measure on its own.

There are ways to guarantee that the object was created only once, such as the One Time Witness . And there are ways to limit or

disable the upgrade of the module, which we will cover in the Package Upgrades chapter.

As follows from the definition, the init function is guaranteed to be called only once when the module is published. So it is a good place to put the code that initializes module's objects and sets up the environment and configuration.

For example, if there's a Capability which is required for certain actions, it should be created in the init function. In the next chapter we will talk about the Capability pattern in more detail.

Next steps

As follows from the definition, the init function is guaranteed to be called only once when the module is published. So it is a good place to put the code that initializes module's objects and sets up the environment and configuration.

For example, if there's a Capability which is required for certain actions, it should be created in the init function. In the next chapter we will talk about the Capability pattern in more detail.