The Move Book

In application design and development, it is often needed to prove publisher authority. This is especially important in the context of digital assets, where the publisher may enable or disable certain features for their assets. The Publisher Object is an object, defined in the Sui Framework , that allows the publisher to prove their authority over a type .

The Publisher object is defined in the sui::package module of the Sui Framework. It is a very simple, non-generic object that can be initialized once per module (and multiple times per package) and is used to prove the authority of the publisher over a type. To claim a Publisher object, the publisher must present a One Time Witness to the package::claim function.

If you're not familiar with the One Time Witness, you can read more about it here .

Here's a simple example of claiming a Publisher object in a module:

The Publisher object has two functions associated with it which are used to prove the publisher's authority over a type:

For small applications or simple use cases, the Publisher object can be used as an admin capability . While in the broader context, the Publisher object has control over system configurations, it can also be used to manage the application's state.

However, Publisher misses some native properties of Capabilities , such as type safety and expressiveness. The signature for the admin_action is not very explicit, can be called by anyone else. And due to Publisher object being standard, there now is a risk of unauthorized access if the from_module check is not performed. So it's important to be cautious when using the Publisher object as an admin role.

Publisher is required for certain features on Sui. Object Display can be created only by the Publisher, and TransferPolicy - an important component of the Kiosk system - also requires the Publisher object to prove ownership of the type.

In the next chapter we will cover the first feature that requires the Publisher object - Object Display - a way to describe objects for clients, and standardize metadata. A must-have for user-friendly applications.

Definition

The Publisher object is defined in the sui::package module of the Sui Framework. It is a very simple, non-generic object that can be initialized once per module (and multiple times per package) and is used to prove the authority of the publisher over a type. To claim a Publisher object, the publisher must present a One Time Witness to the package::claim function.

```
bash // File: sui-framework/sources/package.move public struct Publisher has key, store { id: UID, package: String, module_name: String, }
```

If you're not familiar with the One Time Witness, you can read more about it here .

Here's a simple example of claiming a Publisher object in a module:

```bash module book::publisher;

use sui::package::{Self, Publisher};

/// Some type defined in the module. public struct Book {}

/// The OTW for the module. public struct PUBLISHER has drop {}

/// Uses the One Time Witness to claim the Publisher object. fun init(otw: PUBLISHER, ctx: &mut TxContext) { // Claim the Publisher object. let publisher: Publisher = sui::package::claim(otw, ctx);

// Usually it is transferred to the sender.
// It can also be stored in another object.
transfer::public_transfer(publisher, ctx.sender())

} ```

The Publisher object has two functions associated with it which are used to prove the publisher's authority over a type:

``bash // Checks if the type is from the same module, hence thePublisher` has the // authority over it. assert! (publisher.from_module());
```

// Checks if the type is from the same package, hence the `Publisher` has the // authority over it. assert!(publisher.from_package());
```

For small applications or simple use cases, the Publisher object can be used as an admin capability . While in the broader context, the Publisher object has control over system configurations, it can also be used to manage the application's state.

```bash /// Some action in the application gated by the Publisher object. public fun admin_action(cap: &Publisher, / *app objects...* / param: u64) { assert!(cap.from_module(), ENotAuthorized);

// perform application-specific action

} ```

However, Publisher misses some native properties of Capabilities , such as type safety and expressiveness. The signature for the admin_action is not very explicit, can be called by anyone else. And due to Publisher object being standard, there now is a risk of unauthorized access if the from_module check is not performed. So it's important to be cautious when using the Publisher object as an admin role.

Publisher is required for certain features on Sui. Object Display can be created only by the Publisher, and TransferPolicy - an important component of the Kiosk system - also requires the Publisher object to prove ownership of the type.

In the next chapter we will cover the first feature that requires the Publisher object - Object Display - a way to describe objects for clients, and standardize metadata. A must-have for user-friendly applications.

## Usage

The Publisher object has two functions associated with it which are used to prove the publisher's authority over a type:

``bash // Checks if the type is from the same module, hence thePublisher` has the // authority over it. assert! (publisher.from_module());

// Checks if the type is from the same package, hence the `Publisher` has the // authority over it. assert!(publisher.from_package());
```

For small applications or simple use cases, the Publisher object can be used as an admin capability . While in the broader context, the Publisher object has control over system configurations, it can also be used to manage the application's state.

```bash /// Some action in the application gated by the Publisher object. public fun admin_action(cap: &Publisher, / *app objects...* / param: u64) { assert!(cap.from_module(), ENotAuthorized);

// perform application-specific action

} ```

However, Publisher misses some native properties of Capabilities , such as type safety and expressiveness. The signature for the admin_action is not very explicit, can be called by anyone else. And due to Publisher object being standard, there now is a risk of unauthorized access if the from_module check is not performed. So it's important to be cautious when using the Publisher object as an admin role.

Publisher is required for certain features on Sui. Object Display can be created only by the Publisher, and TransferPolicy - an important component of the Kiosk system - also requires the Publisher object to prove ownership of the type.

In the next chapter we will cover the first feature that requires the Publisher object - Object Display - a way to describe objects for clients, and standardize metadata. A must-have for user-friendly applications.

## Publisher as Admin Role

For small applications or simple use cases, the Publisher object can be used as an admin capability . While in the broader context, the Publisher object has control over system configurations, it can also be used to manage the application's state.

```bash /// Some action in the application gated by the Publisher object. public fun admin_action(cap: &Publisher, / *app objects...* / param: u64) { assert!(cap.from_module(), ENotAuthorized);

// perform application-specific action
```

} ```

However, Publisher misses some native properties of Capabilities , such as type safety and expressiveness. The signature for the admin_action is not very explicit, can be called by anyone else. And due to Publisher object being standard, there now is a risk of unauthorized access if the from_module check is not performed. So it's important to be cautious when using the Publisher object as an admin role.

Publisher is required for certain features on Sui. Object Display can be created only by the Publisher, and TransferPolicy - an important component of the Kiosk system - also requires the Publisher object to prove ownership of the type.

In the next chapter we will cover the first feature that requires the Publisher object - Object Display - a way to describe objects for clients, and standardize metadata. A must-have for user-friendly applications.

Role on Sui

Publisher is required for certain features on Sui. Object Display can be created only by the Publisher, and TransferPolicy - an important component of the Kiosk system - also requires the Publisher object to prove ownership of the type.

In the next chapter we will cover the first feature that requires the Publisher object - Object Display - a way to describe objects for clients, and standardize metadata. A must-have for user-friendly applications.

Next Steps

In the next chapter we will cover the first feature that requires the Publisher object - Object Display - a way to describe objects for clients, and standardize metadata. A must-have for user-friendly applications.