The Move Book

Every module member has a visibility. By default, all module members are private - meaning they are only accessible within the module they are defined in. However, you can add a visibility modifier to make a module member public - visible outside the module, or public(package) - visible in the modules within the same package, or entry - can be called from a transaction but can't be called from other modules.

A function or a struct defined in a module which has no visibility modifier is private to the module. It can't be called from other modules.

The following code will not compile:

Note that just because a struct field is not visible from Move does not mean that its value is kept confidential — it is always possible to read the contents of an on-chain object from outside of Move. You should never store unencrypted secrets inside of objects.

A struct or a function can be made public by adding the public keyword before the fun or struct keyword.

A public function can be imported and called from other modules. The following code will compile:

Unlike some languages, struct fields cannot be made public.

A function with package visibility can be called from any module within the same package, but not from modules in other packages. In other words, it is internal to the package.

A package function can be called from any module within the same package:

Some functions in the framework and standard library are marked with the native modifier. These functions are natively provided by the Move VM and do not have a body in Move source code. To learn more about the native modifier, refer to the Move Reference .

This is an example from std::type_name , learn more about this module in the reflection chapter .

Internal Visibility

A function or a struct defined in a module which has no visibility modifier is private to the module. It can't be called from other modules.

```bash module book::internal_visibility;

// This function can be called from other functions in the same module fun internal() { / ... / }

// Same module -> can call internal() fun call_internal() { internal(); } ```

The following code will not compile:

```bash module book::try_calling_internal;

use book::internal_visibility;

// Different module -> can't call internal() fun try_calling_internal() { internal_visibility::internal(); } ```

Note that just because a struct field is not visible from Move does not mean that its value is kept confidential — it is always possible to read the contents of an on-chain object from outside of Move. You should never store unencrypted secrets inside of objects.

A struct or a function can be made public by adding the public keyword before the fun or struct keyword.

```bash module book::public_visibility;

// This function can be called from other modules public fun public() { / ... / } ```

A public function can be imported and called from other modules. The following code will compile:

```bash module book::try_calling_public {

use book::public_visibility;

// Different module -> can call public() fun try_calling_public() { public_visibility::public(); } ```

Unlike some languages, struct fields cannot be made public.

A function with package visibility can be called from any module within the same package, but not from modules in other packages. In other words, it is internal to the package.

```bash module book::package_visibility;

public(package) fun package_only() { / ... / } ```

A package function can be called from any module within the same package:

```bash module book::try_calling_package;

use book::package_visibility;

// Same package `book` -> can call package_only() fun try_calling_package() { package_visibility::package_only(); } ```

Some functions in the framework and standard library are marked with the native modifier. These functions are natively provided by the Move VM and do not have a body in Move source code. To learn more about the native modifier, refer to the Move Reference .

```bash module std::type_name;

public native fun get(): TypeName; ```

This is an example from std::type_name , learn more about this module in the reflection chapter .

## Public Visibility

A struct or a function can be made public by adding the public keyword before the fun or struct keyword.

```bash module book::public_visibility;

// This function can be called from other modules public fun public() { / ... / } ```

A public function can be imported and called from other modules. The following code will compile:

```bash module book::try_calling_public {

use book::public_visibility;

// Different module -> can call public() fun try_calling_public() { public_visibility::public(); } ```

Unlike some languages, struct fields cannot be made public.

A function with package visibility can be called from any module within the same package, but not from modules in other packages. In other words, it is internal to the package.

```bash module book::package_visibility;

public(package) fun package_only() { / ... / } ```

A package function can be called from any module within the same package:

```bash module book::try_calling_package;

use book::package_visibility;

// Same package `book` -> can call package_only() fun try_calling_package() { package_visibility::package_only(); } ```

Some functions in the framework and standard library are marked with the native modifier. These functions are natively provided by the Move VM and do not have a body in Move source code. To learn more about the native modifier, refer to the Move Reference
```

.

```bash module std::type_name;

public native fun get(): TypeName; ```

This is an example from std::type_name , learn more about this module in the reflection chapter .

## Package Visibility

A function with package visibility can be called from any module within the same package, but not from modules in other packages. In other words, it is internal to the package.

```bash module book::package_visibility;

public(package) fun package_only() { / ... / } ```

A package function can be called from any module within the same package:

```bash module book::try_calling_package;

use book::package_visibility;

// Same package `book` -> can call package_only() fun try_calling_package() { package_visibility::package_only(); } ```

Some functions in the framework and standard library are marked with the native modifier. These functions are natively provided by the Move VM and do not have a body in Move source code. To learn more about the native modifier, refer to the Move Reference .

```bash module std::type_name;

public native fun get(): TypeName; ```

This is an example from std::type_name , learn more about this module in the reflection chapter .

## Native Functions

Some functions in the framework and standard library are marked with the native modifier. These functions are natively provided by the Move VM and do not have a body in Move source code. To learn more about the native modifier, refer to the Move Reference .

```bash module std::type_name;

public native fun get(): TypeName; ```

This is an example from std::type_name , learn more about this module in the reflection chapter .

## Further Reading
```