The Move Book

Move 2024 is the new edition of the Move language that is maintained by Mysten Labs. This guide is intended to help you understand the differences between the 2024 edition and the previous version of the Move language.

This guide provides a high-level overview of the changes in the new edition. For a more detailed and exhaustive list of changes, refer to the Sui Documentation .

To use the new edition, you need to specify the edition in the move file. The edition is specified in the move file using the edition keyword. Currently, the only available edition is 2024.beta .

The Move CLI has a migration tool that updates the code to the new edition. To use the migration tool, run the following command:

The migration tool will update the code to use the let mut syntax, the new public modifier for structs, and the public(package) function visibility instead of friend declarations.

Move 2024 introduces let mut syntax to declare mutable variables. The let mut syntax is used to declare a mutable variable that can be changed after it is declared.

let mut declaration is now required for mutable variables. Compiler will emit an error if you try to reassign a variable without the mut keyword.

Additionally, the mut keyword is used in tuple destructuring and function arguments to declare mutable variables.

In Move 2024, the friend keyword is deprecated. Instead, you can use the public(package) visibility modifier to make functions visible to other modules in the same package.

In Move 2024, structs get a visibility modifier. Currently, the only available visibility modifier is public .

In the new edition, functions which have a struct as the first argument are associated with the struct. This means that the function can be called using the dot notation. Methods defined in the same module with the type are automatically exported.

Methods are automatically exported if the type is defined in the same module as the method. It is impossible to export methods for types defined in other modules. However, you can create custom aliases for methods in the module scope.

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

Using the New Edition

To use the new edition, you need to specify the edition in the move file. The edition is specified in the move file using the edition keyword. Currently, the only available edition is 2024.beta .

```bash edition = "2024"

# alternatively, for new features:

edition = "2024.beta" ```

The Move CLI has a migration tool that updates the code to the new edition. To use the migration tool, run the following command:
```

```
bash $ sui move migrate
```

The migration tool will update the code to use the let mut syntax, the new public modifier for structs, and the public(package) function visibility instead of friend declarations.

Move 2024 introduces let mut syntax to declare mutable variables. The let mut syntax is used to declare a mutable variable that can be changed after it is declared.

let mut declaration is now required for mutable variables. Compiler will emit an error if you try to reassign a variable without the mut keyword.

```bash // Move 2020 let x: u64 = 10; x = 20;

// Move 2024 let mut x: u64 = 10; x = 20; ```

Additionally, the mut keyword is used in tuple destructuring and function arguments to declare mutable variables.

```bash // takes by value and mutates fun takes_by_value_and_mutates(mut v: Value): Value { v.field = 10; v }

// `mut` should be placed before the variable name fun destruct() { let (x, y) = point::get_point(); let (mut x, y) = point::get_point(); let (mut x, mut y) = point::get_point(); }

// in struct unpack fun unpack() { let Point { x, mut y } = point::get_point(); let Point { mut x, mut y } = point::get_point(); } ```

In Move 2024, the friend keyword is deprecated. Instead, you can use the public(package) visibility modifier to make functions visible to other modules in the same package.

```bash // Move 2020 friend book::friend_module; public(friend) fun protected_function() {}

// Move 2024 public(package) fun protected_function_2024() {} ```

In Move 2024, structs get a visibility modifier. Currently, the only available visibility modifier is public .

```bash // Move 2020 struct Book {}

// Move 2024 public struct Book {} ```

In the new edition, functions which have a struct as the first argument are associated with the struct. This means that the function can be called using the dot notation. Methods defined in the same module with the type are automatically exported.

Methods are automatically exported if the type is defined in the same module as the method. It is impossible to export methods for types defined in other modules. However, you can create custom aliases for methods in the module scope.

```bash public fun count(c: &Counter): u64 { / ... / }

fun use_counter() { // move 2020 let count = counter::count(&c);

```
// move 2024
let count = c.count();
```

} ```

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

```bash fun aliases() { // vector to string and ascii string let str: String = b"Hello, World!".to_string(); let ascii: ascii::String = b"Hello, World!".to_ascii_string();

```
// address to bytes
let bytes = @0xa11ce.to_bytes();
```

} ```

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let
first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls
*vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

## Migration Tool

The Move CLI has a migration tool that updates the code to the new edition. To use the migration tool, run the following command:

```
bash $ sui move migrate
```

The migration tool will update the code to use the let mut syntax, the new public modifier for structs, and the public(package) function visibility instead of friend declarations.

Move 2024 introduces let mut syntax to declare mutable variables. The let mut syntax is used to declare a mutable variable that can be changed after it is declared.

let mut declaration is now required for mutable variables. Compiler will emit an error if you try to reassign a variable without the mut keyword.

```bash // Move 2020 let x: u64 = 10; x = 20;

// Move 2024 let mut x: u64 = 10; x = 20; ```

Additionally, the mut keyword is used in tuple destructuring and function arguments to declare mutable variables.

```bash // takes by value and mutates fun takes_by_value_and_mutates(mut v: Value): Value { v.field = 10; v }

// `mut` should be placed before the variable name fun destruct() { let (x, y) = point::get_point(); let (mut x, y) = point::get_point(); let (mut x, mut y) = point::get_point(); }

// in struct unpack fun unpack() { let Point { x, mut y } = point::get_point(); let Point { mut x, mut y } = point::get_point(); } ```

In Move 2024, the friend keyword is deprecated. Instead, you can use the public(package) visibility modifier to make functions visible to other modules in the same package.

```bash // Move 2020 friend book::friend_module; public(friend) fun protected_function() {}

// Move 2024 public(package) fun protected_function_2024() {} ```

In Move 2024, structs get a visibility modifier. Currently, the only available visibility modifier is public .

```bash // Move 2020 struct Book {}

// Move 2024 public struct Book {} ```

In the new edition, functions which have a struct as the first argument are associated with the struct. This means that the function can be called using the dot notation. Methods defined in the same module with the type are automatically exported.

Methods are automatically exported if the type is defined in the same module as the method. It is impossible to export methods for types defined in other modules. However, you can create custom aliases for methods in the module scope.

```bash public fun count(c: &Counter): u64 { / ... / }

fun use_counter() { // move 2020 let count = counter::count(&c);

```
// move 2024
let count = c.count();
```

} ```

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

```bash fun aliases() { // vector to string and ascii string let str: String = b"Hello, World!".to_string(); let ascii: ascii::String = b"Hello, World!".to_ascii_string();

```
// address to bytes
let bytes = @0xa11ce.to_bytes();
```

} ```

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let
first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls
*vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

## Mutable bindings with

Move 2024 introduces let mut syntax to declare mutable variables. The let mut syntax is used to declare a mutable variable that can be changed after it is declared.

let mut declaration is now required for mutable variables. Compiler will emit an error if you try to reassign a variable without the mut keyword.

```bash // Move 2020 let x: u64 = 10; x = 20;

// Move 2024 let mut x: u64 = 10; x = 20; ```

Additionally, the mut keyword is used in tuple destructuring and function arguments to declare mutable variables.

```bash // takes by value and mutates fun takes_by_value_and_mutates(mut v: Value): Value { v.field = 10; v }

// mut should be placed before the variable name fun destruct() { let (x, y) = point::get_point(); let (mut x, y) = point::get_point(); let (mut x, mut y) = point::get_point(); }

// in struct unpack fun unpack() { let Point { x, mut y } = point::get_point(); let Point { mut x, mut y } = point::get_point(); } ```

In Move 2024, the friend keyword is deprecated. Instead, you can use the public(package) visibility modifier to make functions visible to other modules in the same package.

```bash // Move 2020 friend book::friend_module; public(friend) fun protected_function() {}

// Move 2024 public(package) fun protected_function_2024() {} ```

In Move 2024, structs get a visibility modifier. Currently, the only available visibility modifier is public .

```bash // Move 2020 struct Book {}

// Move 2024 public struct Book {} ```

In the new edition, functions which have a struct as the first argument are associated with the struct. This means that the function can be called using the dot notation. Methods defined in the same module with the type are automatically exported.

Methods are automatically exported if the type is defined in the same module as the method. It is impossible to export methods for types defined in other modules. However, you can create custom aliases for methods in the module scope.

```bash public fun count(c: &Counter): u64 { / ... / }

fun use_counter() { // move 2020 let count = counter::count(&c);

```
// move 2024
let count = c.count();
```

} ```

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

```bash fun aliases() { // vector to string and ascii string let str: String = b"Hello, World!".to_string(); let ascii: ascii::String = b"Hello, World!".to_ascii_string();

```
// address to bytes
let bytes = @0xa11ce.to_bytes();
```

} ```

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let
first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls
*vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

## Friends are Deprecated

In Move 2024, the friend keyword is deprecated. Instead, you can use the public(package) visibility modifier to make functions visible to other modules in the same package.

```bash // Move 2020 friend book::friend_module; public(friend) fun protected_function() {}

// Move 2024 public(package) fun protected_function_2024() {} ```

In Move 2024, structs get a visibility modifier. Currently, the only available visibility modifier is public .

```bash // Move 2020 struct Book {}

// Move 2024 public struct Book {} ```

In the new edition, functions which have a struct as the first argument are associated with the struct. This means that the function can be called using the dot notation. Methods defined in the same module with the type are automatically exported.

Methods are automatically exported if the type is defined in the same module as the method. It is impossible to export methods for types defined in other modules. However, you can create custom aliases for methods in the module scope.

```bash public fun count(c: &Counter): u64 { / ... / }

fun use_counter() { // move 2020 let count = counter::count(&c);

```
// move 2024
let count = c.count();
```

} ```

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

```bash fun aliases() { // vector to string and ascii string let str: String = b"Hello, World!".to_string(); let ascii: ascii::String = b"Hello, World!".to_ascii_string();

```
// address to bytes
let bytes = @0xa11ce.to_bytes();
```

} ```

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let
first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls
*vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

## Struct Visibility

In Move 2024, structs get a visibility modifier. Currently, the only available visibility modifier is public .

```bash // Move 2020 struct Book {}

// Move 2024 public struct Book {} ```

In the new edition, functions which have a struct as the first argument are associated with the struct. This means that the function can be called using the dot notation. Methods defined in the same module with the type are automatically exported.

Methods are automatically exported if the type is defined in the same module as the method. It is impossible to export methods for types defined in other modules. However, you can create custom aliases for methods in the module scope.

```bash public fun count(c: &Counter): u64 { / ... / }

fun use_counter() { // move 2020 let count = counter::count(&c);

```
// move 2024
let count = c.count();
```

} ```

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

```bash fun aliases() { // vector to string and ascii string let str: String = b"Hello, World!".to_string(); let ascii: ascii::String = b"Hello, World!".to_ascii_string();

```
// address to bytes
let bytes = @0xa11ce.to_bytes();
```

} ```

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .
```

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let
first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls
*vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

## Method Syntax

In the new edition, functions which have a struct as the first argument are associated with the struct. This means that the function can be called using the dot notation. Methods defined in the same module with the type are automatically exported.

Methods are automatically exported if the type is defined in the same module as the method. It is impossible to export methods for types defined in other modules. However, you can create custom aliases for methods in the module scope.

```bash public fun count(c: &Counter): u64 { / ... / }

fun use_counter() { // move 2020 let count = counter::count(&c);

```
// move 2024
let count = c.count();
```

} ```

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

```bash fun aliases() { // vector to string and ascii string let str: String = b"Hello, World!".to_string(); let ascii: ascii::String = b"Hello, World!".to_ascii_string();

```
// address to bytes
let bytes = @0xa11ce.to_bytes();
```

} ```

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let
first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls
*vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

## Methods for Built-in Types

In Move 2024, some of the native and standard types received associated methods. For example, the vector type has a to_string method that converts the vector into a UTF8 string.

```bash fun aliases() { // vector to string and ascii string let str: String = b"Hello, World!".to_string(); let ascii: ascii::String = b"Hello, World!".to_ascii_string();

```
// address to bytes
let bytes = @0xa11ce.to_bytes();
```

} ```

For the full list of built-in aliases, refer to the Standard Library and Sui Framework source code.

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls *vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;
```

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

Borrowing Operator

Some of the built-in types support borrowing operators. The borrowing operator is used to get a reference to the element at the specified index. The borrowing operator is defined as [] .

```
bash fun play_vec() { let v = vector[1,2,3,4]; let first = &v[0]; // calls vector::borrow(v, 0) let first_mut = &mut v[0]; // calls vector::borrow_mut(v, 0) let first_copy = v[0]; // calls *vector::borrow(v, 0) }
```

Types that support the borrowing operator are:

To implement the borrowing operator for a custom type, you need to add a #[syntax(index)] attribute to the methods.

```bash

# [syntax(index)]

public fun borrow(c: &List, key: String): &T { / ... / }

# [syntax(index)]

public fun borrow_mut(c: &mut List, key: String): &mut T { / ... / } ```

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```

## Method Aliases

In Move 2024, methods can be associated with types. The alias can be defined for any type locally to the module; or publicly, if the type is defined in the same module.

```bash // my_module.move // Local: type is foreign to the module use fun my_custom_function as vector.do_magic;

// sui-framework/kiosk/kiosk.move // Exported: type is defined in the same module public use fun kiosk_owner_cap_for as KioskOwnerCap.kiosk; ```
```