Immutable Objects

Objects in Sui can have different types of ownership, with two broad categories: immutable objects and mutable objects. An immutable object is an object that can't be mutated, transferred, or deleted. Immutable objects have no owner, so anyone can use them.

To convert an object into an immutable object, call the public_freeze_object function from the transfer module :

This call makes the specified object immutable. This is a non-reversible operation. You should freeze an object only when you are certain that you don't need to mutate it.

You can see this function's use in one of the color_object example module tests. The test creates a new (owned) ColorObject , then calls public_freeze_object to turn it into an immutable object.

In the preceding test, you must already own a ColorObject to pass it in. At the end of this call, this object is frozen and can never be mutated. It is also no longer owned by anyone.

The transfer::public_freeze_object function requires that you pass the object by value. If you are allowed to pass the object by a mutable reference, you could still mutate the object after the public_freeze_object call. This contradicts the fact that it should have become immutable.

Alternatively, you can also provide an API that creates an immutable object at creation:

This function creates a new ColorObject and immediately makes it immutable before it has an owner.

After an object becomes immutable, the rules of who can use this object in Sui Move calls change: You can only pass an immutable object as a read-only, immutable reference to Sui Move entry functions as &T .

Anyone can use immutable objects.

Consider a function that copies the value of one object to another:

In this function, anyone can pass an immutable object as the first argument, from , but not the second argument. Because you can never mutate immutable objects, there's no data race, even when multiple transactions are using the same immutable object at the same time. Hence, the existence of immutable objects does not pose any requirement on consensus.

You can interact with immutable objects in unit tests using test_scenario::take_immutable to take an immutable object wrapper from global storage, and test_scenario::return_immutable to return the wrapper back to the global storage.

The test_scenario::take_immutable function is required because you can access immutable objects solely through read-only references. The test_scenario runtime keeps track of the usage of this immutable object. If the compiler does not return the object via test_scenario::return_immutable before the start of the next transaction, the test stops.

To see it work in action:

This test submits a transaction as sender1 , which tries to create an immutable object.

The has_most_recent_for_sender function no longer returns true , because the object is no longer owned. To take this object:

To show that this object is indeed not owned by anyone, start the next transaction with sender2 . Note that it used take_immutable and succeeded. This means that any sender can take an immutable object. To return the object, call the return_immutable function.

To examine whether this object is immutable, add a function that tries to mutate a ColorObject :

As you have learned, the function fails when the ColorObject is immutable.

First, view the objects you own:

Publish the ColorObject code on-chain using the Sui Client CLI:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Set the package object ID to the $PACKAGE environment variable, if you have it set. Then create a new ColorObject :

Set the newly created object ID to $OBJECT . To view the objects in the current active address:

You should see an object with the ID you used for $OBJECT . To turn it into an immutable object:

View the list of objects again:

$OBJECT is no longer listed. It's no longer owned by anyone. You can see that it's now immutable by querying the object information:

The response includes:

If you try to mutate it:

The response indicates that you can't pass an immutable object to a mutable argument.

Create immutable object

To convert an object into an immutable object, call the public_freeze_object function from the transfer module :

This call makes the specified object immutable. This is a non-reversible operation. You should freeze an object only when you are certain that you don't need to mutate it.

You can see this function's use in one of the color_object example module tests. The test creates a new (owned) ColorObject , then calls public_freeze_object to turn it into an immutable object.

In the preceding test, you must already own a ColorObject to pass it in. At the end of this call, this object is frozen and can never be mutated. It is also no longer owned by anyone.

The transfer::public_freeze_object function requires that you pass the object by value. If you are allowed to pass the object by a mutable reference, you could still mutate the object after the public_freeze_object call. This contradicts the fact that it should have become immutable.

Alternatively, you can also provide an API that creates an immutable object at creation:

This function creates a new ColorObject and immediately makes it immutable before it has an owner.

After an object becomes immutable, the rules of who can use this object in Sui Move calls change: You can only pass an immutable object as a read-only, immutable reference to Sui Move entry functions as &T .

Anyone can use immutable objects.

Consider a function that copies the value of one object to another:

In this function, anyone can pass an immutable object as the first argument, from , but not the second argument. Because you can never mutate immutable objects, there's no data race, even when multiple transactions are using the same immutable object at the same time. Hence, the existence of immutable objects does not pose any requirement on consensus.

You can interact with immutable objects in unit tests using test_scenario::take_immutable to take an immutable object wrapper from global storage, and test_scenario::return_immutable to return the wrapper back to the global storage.

The test_scenario::take_immutable function is required because you can access immutable objects solely through read-only references. The test_scenario runtime keeps track of the usage of this immutable object. If the compiler does not return the object via test_scenario::return_immutable before the start of the next transaction, the test stops.

To see it work in action:

This test submits a transaction as sender1 , which tries to create an immutable object.

The has_most_recent_for_sender function no longer returns true , because the object is no longer owned. To take this object:

To show that this object is indeed not owned by anyone, start the next transaction with sender2 . Note that it used take_immutable and succeeded. This means that any sender can take an immutable object. To return the object, call the return_immutable function.

To examine whether this object is immutable, add a function that tries to mutate a ColorObject :

As you have learned, the function fails when the ColorObject is immutable.

First, view the objects you own:

Publish the ColorObject code on-chain using the Sui Client CLI:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Set the package object ID to the $PACKAGE environment variable, if you have it set. Then create a new ColorObject :

Set the newly created object ID to $OBJECT . To view the objects in the current active address:

You should see an object with the ID you used for $OBJECT . To turn it into an immutable object:

View the list of objects again:

$OBJECT is no longer listed. It's no longer owned by anyone. You can see that it's now immutable by querying the object information:

The response includes:

If you try to mutate it:

The response indicates that you can't pass an immutable object to a mutable argument.

Use immutable object

After an object becomes immutable, the rules of who can use this object in Sui Move calls change: You can only pass an immutable object as a read-only, immutable reference to Sui Move entry functions as &T .

Anyone can use immutable objects.

Consider a function that copies the value of one object to another:

In this function, anyone can pass an immutable object as the first argument, from , but not the second argument. Because you can never mutate immutable objects, there's no data race, even when multiple transactions are using the same immutable object at the same time. Hence, the existence of immutable objects does not pose any requirement on consensus.

You can interact with immutable objects in unit tests using test_scenario::take_immutable to take an immutable object wrapper from global storage, and test_scenario::return_immutable to return the wrapper back to the global storage.

The test_scenario::take_immutable function is required because you can access immutable objects solely through read-only references. The test_scenario runtime keeps track of the usage of this immutable object. If the compiler does not return the object via test_scenario::return_immutable before the start of the next transaction, the test stops.

To see it work in action:

This test submits a transaction as sender1 , which tries to create an immutable object.

The has_most_recent_for_sender function no longer returns true , because the object is no longer owned. To take this object:

To show that this object is indeed not owned by anyone, start the next transaction with sender2 . Note that it used take_immutable and succeeded. This means that any sender can take an immutable object. To return the object, call the return_immutable function.

To examine whether this object is immutable, add a function that tries to mutate a ColorObject :

As you have learned, the function fails when the ColorObject is immutable.

First, view the objects you own:

Publish the ColorObject code on-chain using the Sui Client CLI:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Set the package object ID to the $PACKAGE environment variable, if you have it set. Then create a new ColorObject :

Set the newly created object ID to $OBJECT . To view the objects in the current active address:

You should see an object with the ID you used for $OBJECT . To turn it into an immutable object:

View the list of objects again:

$OBJECT is no longer listed. It's no longer owned by anyone. You can see that it's now immutable by querying the object information:

The response includes:

If you try to mutate it:

The response indicates that you can't pass an immutable object to a mutable argument.

Test immutable object

You can interact with immutable objects in unit tests using test_scenario::take_immutable to take an immutable object wrapper from global storage, and test_scenario::return_immutable to return the wrapper back to the global storage.

The test_scenario::take_immutable function is required because you can access immutable objects solely through read-only references. The test_scenario runtime keeps track of the usage of this immutable object. If the compiler does not return the object via test_scenario::return_immutable before the start of the next transaction, the test stops.

To see it work in action:

This test submits a transaction as sender1 , which tries to create an immutable object.

The has_most_recent_for_sender function no longer returns true , because the object is no longer owned. To take this object:

To show that this object is indeed not owned by anyone, start the next transaction with sender2 . Note that it used take_immutable and succeeded. This means that any sender can take an immutable object. To return the object, call the return_immutable function.

To examine whether this object is immutable, add a function that tries to mutate a ColorObject :

As you have learned, the function fails when the ColorObject is immutable.

First, view the objects you own:

Publish the ColorObject code on-chain using the Sui Client CLI:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Set the package object ID to the $PACKAGE environment variable, if you have it set. Then create a new ColorObject :

Set the newly created object ID to $OBJECT . To view the objects in the current active address:

You should see an object with the ID you used for $OBJECT . To turn it into an immutable object:

View the list of objects again:

$OBJECT is no longer listed. It's no longer owned by anyone. You can see that it's now immutable by querying the object information:

The response includes:

If you try to mutate it:

The response indicates that you can't pass an immutable object to a mutable argument.

On-chain interactions

First, view the objects you own:

Publish the ColorObject code on-chain using the Sui Client CLI:

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Set the package object ID to the $PACKAGE environment variable, if you have it set. Then create a new ColorObject :

Set the newly created object ID to $OBJECT . To view the objects in the current active address:

You should see an object with the ID you used for $OBJECT . To turn it into an immutable object:

View the list of objects again:

$OBJECT is no longer listed. It's no longer owned by anyone. You can see that it's now immutable by querying the object information:

The response includes:

If you try to mutate it:

The response indicates that you can't pass an immutable object to a mutable argument.