Move Concepts

Move is an open source language for writing safe packages to manipulate on-chain objects (sometimes referred to as "smart contracts"). Move is a platform-agnostic language to enable common libraries, tooling, and developer communities across blockchains with vastly different data and execution models. Move is adaptable to meet the needs of the blockchain the code operates on, see Move on Sui to review enhancements made to Move for optimization on the Sui blockchain.

You can use Move to define, create, and manage programmable Sui objects representing user-level assets. Sui's object system is implemented by adding new functionality to Move while also imposing additional restrictions. See Object Model for more details.

Move on Sui contains some important differences from Move on other blockchains. Sui takes advantage of Move's security and flexibility and enhances it with the features to vastly improve throughput, reduce delays in finality, and make Move programming more approachable. For full details, see the Sui Smart Contracts Platform whitepaper.

Where the Sui documentation refers to the Move language, the content is documenting the specific Move implementation on the Sui blockchain. If relevant, the documentation expressly refers to the original use case for the Move language as Move on Diem.

Key differences with Move on Sui include:

In Move on Diem, global storage is part of the programming model. Resources and modules are held in global storage, owned by an account which has an address. Transactions are free to access resources from any account in global storage when they run, using special operations such as move_to and move_from .

This approach introduces a scaling issue, as it is not possible to statically determine which transactions are contending over the same resource and which are not. This is similar to the scaling issues faced by other blockchains where smart contracts typically store account information in large, internal mappings, which limit throughput.

Move on Sui addresses the scaling issue by not having global storage, or its related operations. When objects (in contrast to resources) and packages (sets of modules) are stored on Sui, they are each given unique identifiers. All a transaction's inputs are explicitly specified up-front using these unique identifiers, to allow the chain to schedule transactions with non-overlapping inputs in parallel.

In Move on Diem, there is a 16-byte address type used to represent account addresses in global storage. A 16 byte address is sufficient for the Move on Diem security model.

Sui doesn't have global storage, so address is re-purposed as a 32-byte identifier used for both objects and accounts. Each transaction is signed by an account (the "sender") that is accessible from the transaction context, and each object stores its address wrapped in its id: UID field.

See Address in The Move Book for an overview on addresses and refer to object.move in the Sui Framework for implementation details.

In Move on Diem, the key ability indicates that the type is a resource, meaning it (along with an account address) can be used as a key in global storage.

On Sui, the key ability indicates that a struct is an object type and comes with an additional requirement that the first field of the struct has signature id: UID , to contain the object's unique address on-chain. Sui's bytecode verifier ensures that new objects are always assigned fresh UID s (identifiers are never re-used).

As described in Object-centric global storage , you publish Move modules into Sui storage. The Sui runtime executes a special initializer function you optionally define in a module only once at the time of module publication to pre-initialize module-specific data (for example, creating singleton objects). See Module Initializer in The Move Book for more information.

The entry keyword has a specific use case in Move on Sui. Use entry when you want some functionality to be usable by anyone on chain, but not be wrapped around other Move logic. In other words, a function can be called in PTBs but not in other Sui packages. For example, this is utilized when using Sui's on-chain randomness standard to prevent other smart contract engineers from creating logic to essentially front- or back-run the randomness generation. Find more information about this in the on-chain randomness page .

Make your function private (don't add the public visibility keyword) and mark it with the entry keyword, as in entry fun example_function .

In addition to this Sui-specific use case, there are other rules and restrictions for entry functions.

To learn more about using Move on Sui, see the following sites:

Move on Sui

Move on Sui contains some important differences from Move on other blockchains. Sui takes advantage of Move's security and flexibility and enhances it with the features to vastly improve throughput, reduce delays in finality, and make Move programming more approachable. For full details, see the Sui Smart Contracts Platform whitepaper.

Where the Sui documentation refers to the Move language, the content is documenting the specific Move implementation on the Sui blockchain. If relevant, the documentation expressly refers to the original use case for the Move language as Move on Diem.

Key differences with Move on Sui include:

In Move on Diem, global storage is part of the programming model. Resources and modules are held in global storage, owned by an account which has an address. Transactions are free to access resources from any account in global storage when they run, using special operations such as move_to and move_from .

This approach introduces a scaling issue, as it is not possible to statically determine which transactions are contending over the same resource and which are not. This is similar to the scaling issues faced by other blockchains where smart contracts typically store account information in large, internal mappings, which limit throughput.

Move on Sui addresses the scaling issue by not having global storage, or its related operations. When objects (in contrast to resources) and packages (sets of modules) are stored on Sui, they are each given unique identifiers. All a transaction's inputs are explicitly specified up-front using these unique identifiers, to allow the chain to schedule transactions with non-overlapping inputs in parallel.

In Move on Diem, there is a 16-byte address type used to represent account addresses in global storage. A 16 byte address is sufficient for the Move on Diem security model.

Sui doesn't have global storage, so address is re-purposed as a 32-byte identifier used for both objects and accounts. Each transaction is signed by an account (the "sender") that is accessible from the transaction context, and each object stores its address wrapped in its id: UID field.

See Address in The Move Book for an overview on addresses and refer to object.move in the Sui Framework for implementation details.

In Move on Diem, the key ability indicates that the type is a resource, meaning it (along with an account address) can be used as a key in global storage.

On Sui, the key ability indicates that a struct is an object type and comes with an additional requirement that the first field of the struct has signature id: UID , to contain the object's unique address on-chain. Sui's bytecode verifier ensures that new objects are always assigned fresh UID s (identifiers are never re-used).

As described in Object-centric global storage , you publish Move modules into Sui storage. The Sui runtime executes a special initializer function you optionally define in a module only once at the time of module publication to pre-initialize module-specific data (for example, creating singleton objects). See Module Initializer in The Move Book for more information.

The entry keyword has a specific use case in Move on Sui. Use entry when you want some functionality to be usable by anyone on chain, but not be wrapped around other Move logic. In other words, a function can be called in PTBs but not in other Sui packages. For example, this is utilized when using Sui's on-chain randomness standard to prevent other smart contract engineers from creating logic to essentially front- or back-run the randomness generation. Find more information about this in the on-chain randomness page .

Make your function private (don't add the public visibility keyword) and mark it with the entry keyword, as in entry fun example_function .

In addition to this Sui-specific use case, there are other rules and restrictions for entry functions.

To learn more about using Move on Sui, see the following sites:

Key differences

Key differences with Move on Sui include:

In Move on Diem, global storage is part of the programming model. Resources and modules are held in global storage, owned by an account which has an address. Transactions are free to access resources from any account in global storage when they run, using special operations such as move_to and move_from .

This approach introduces a scaling issue, as it is not possible to statically determine which transactions are contending over the same resource and which are not. This is similar to the scaling issues faced by other blockchains where smart contracts typically store account information in large, internal mappings, which limit throughput.

Move on Sui addresses the scaling issue by not having global storage, or its related operations. When objects (in contrast to resources) and packages (sets of modules) are stored on Sui, they are each given unique identifiers. All a transaction's inputs are explicitly specified up-front using these unique identifiers, to allow the chain to schedule transactions with non-overlapping inputs in parallel.

In Move on Diem, there is a 16-byte address type used to represent account addresses in global storage. A 16 byte address is sufficient for the Move on Diem security model.

Sui doesn't have global storage, so address is re-purposed as a 32-byte identifier used for both objects and accounts. Each transaction is signed by an account (the "sender") that is accessible from the transaction context, and each object stores its address wrapped in its id: UID field.

See Address in The Move Book for an overview on addresses and refer to object.move in the Sui Framework for implementation details.

In Move on Diem, the key ability indicates that the type is a resource, meaning it (along with an account address) can be used as a key in global storage.

On Sui, the key ability indicates that a struct is an object type and comes with an additional requirement that the first field of the struct has signature id: UID , to contain the object's unique address on-chain. Sui's bytecode verifier ensures that new objects are always assigned fresh UID s (identifiers are never re-used).

As described in Object-centric global storage , you publish Move modules into Sui storage. The Sui runtime executes a special initializer function you optionally define in a module only once at the time of module publication to pre-initialize module-specific data (for example, creating singleton objects). See Module Initializer in The Move Book for more information.

The entry keyword has a specific use case in Move on Sui. Use entry when you want some functionality to be usable by anyone on chain, but not be wrapped around other Move logic. In other words, a function can be called in PTBs but not in other Sui packages. For example, this is utilized when using Sui's on-chain randomness standard to prevent other smart contract engineers from creating logic to essentially front- or back-run the randomness generation. Find more information about this in the on-chain randomness page .

Make your function private (don't add the public visibility keyword) and mark it with the entry keyword, as in entry fun example_function .

In addition to this Sui-specific use case, there are other rules and restrictions for entry functions.

To learn more about using Move on Sui, see the following sites:

Related links

To learn more about using Move on Sui, see the following sites: