Closed-Loop Token

Using the Closed-Loop Token standard, you can limit the applications that can use the token and set up custom policies for transfers, spends, and conversions. The sui::token module in the Sui framework defines the standard.

The Coin standard on Sui is an example of an open-loop system - coins are free-flowing, wrappable , freely transferable and you can store them in any application. The best real world analogy would be cash - hardly regulated and can be freely used and passed.

Some applications, however, require constraining the scope of the token to a specific purpose. For example, some applications might need a token that you can only use for a specific service, or that an authorized account can only use, or a token that you can block certain accounts from using. A real-world analogy would be a bank account - regulated, bank-controlled, and compliant with certain rules and policies.

Unlike Coin, which has key + store abilities and thus supports wrapping and public transfers, Token has only the key ability and cannot be wrapped, stored as a dynamic field, or freely transferred (unless there's a custom policy for that). Due to this restriction, Token can only be owned by an account and can't be stored in an application (however, it can be "spent" - see Spending section section).

You can set up any rules for transfers, spends, and conversions for the tokens you create. You specify these rules per action in the TokenPolicy . Rules are custom programmable restrictions that you can use to implement any request authorization or validation logic.

For example, a policy can set a limit on a transfer - X tokens per operation; or require user verification before spending tokens; or allow spending tokens only on a specific service.

You can reuse rules across different policies and applications; and you can freely combine rules to create complex policies.

Tokens have a set of public and protected actions that you can use to manage the token. Public actions are available to everyone and don't require any authorization. They have similar APIs to coins, but operate on the Token type:

See Coin Token Comparison for coin and token methods comparison.

Protected actions are ones that issue an ActionRequest - a hot-potato struct that must be resolved for the transaction to succeed. There are three main ways to resolve an ActionRequest , most common of which is via the TokenPolicy .

The previous methods are included in the base implementation, however it is possible to create ActionRequest s for custom actions.

Protected actions are disabled by default but you can enable them in a TokenPolicy . Additionally, you can set custom restrictions called rules that a specific action must satisfy for it to succeed.

Background and use cases

The Coin standard on Sui is an example of an open-loop system - coins are free-flowing, wrappable , freely transferable and you can store them in any application. The best real world analogy would be cash - hardly regulated and can be freely used and passed.

Some applications, however, require constraining the scope of the token to a specific purpose. For example, some applications might need a token that you can only use for a specific service, or that an authorized account can only use, or a token that you can block certain accounts from using. A real-world analogy would be a bank account - regulated, bank-controlled, and compliant with certain rules and policies.

Unlike Coin, which has key + store abilities and thus supports wrapping and public transfers, Token has only the key ability and cannot be wrapped, stored as a dynamic field, or freely transferred (unless there's a custom policy for that). Due to this restriction, Token can only be owned by an account and can't be stored in an application (however, it can be "spent" - see Spending section section).

You can set up any rules for transfers, spends, and conversions for the tokens you create. You specify these rules per action in the TokenPolicy . Rules are custom programmable restrictions that you can use to implement any request authorization or validation logic.

For example, a policy can set a limit on a transfer - X tokens per operation; or require user verification before spending tokens; or allow spending tokens only on a specific service.

You can reuse rules across different policies and applications; and you can freely combine rules to create complex policies.

Tokens have a set of public and protected actions that you can use to manage the token. Public actions are available to everyone and don't require any authorization. They have similar APIs to coins, but operate on the Token type:

See Coin Token Comparison for coin and token methods comparison.

Protected actions are ones that issue an ActionRequest - a hot-potato struct that must be resolved for the transaction to succeed. There are three main ways to resolve an ActionRequest , most common of which is via the TokenPolicy .

The previous methods are included in the base implementation, however it is possible to create ActionRequest s for custom actions.

Protected actions are disabled by default but you can enable them in a TokenPolicy . Additionally, you can set custom restrictions called rules that a specific action must satisfy for it to succeed.

Difference with Coin

Unlike Coin, which has key + store abilities and thus supports wrapping and public transfers, Token has only the key ability and cannot be wrapped, stored as a dynamic field, or freely transferred (unless there's a custom policy for that). Due to this restriction, Token can only be owned by an account and can't be stored in an application (however, it can be "spent" - see Spending section section).

You can set up any rules for transfers, spends, and conversions for the tokens you create. You specify these rules per action in the TokenPolicy . Rules are custom programmable restrictions that you can use to implement any request authorization or validation logic.

For example, a policy can set a limit on a transfer - X tokens per operation; or require user verification before spending tokens; or allow spending tokens only on a specific service.

You can reuse rules across different policies and applications; and you can freely combine rules to create complex policies.

Tokens have a set of public and protected actions that you can use to manage the token. Public actions are available to everyone and don't require any authorization. They have similar APIs to coins, but operate on the Token type:

See Coin Token Comparison for coin and token methods comparison.

Protected actions are ones that issue an ActionRequest - a hot-potato struct that must be resolved for the transaction to succeed. There are three main ways to resolve an ActionRequest , most common of which is via the TokenPolicy .

The previous methods are included in the base implementation, however it is possible to create ActionRequest s for custom actions.

Protected actions are disabled by default but you can enable them in a TokenPolicy . Additionally, you can set custom restrictions called rules that a specific action must satisfy for it to succeed.

Compliance and rules

You can set up any rules for transfers, spends, and conversions for the tokens you create. You specify these rules per action in the TokenPolicy . Rules are custom programmable restrictions that you can use to implement any request authorization or validation logic.

For example, a policy can set a limit on a transfer - X tokens per operation; or require user verification before spending tokens; or allow spending tokens only on a specific service.

You can reuse rules across different policies and applications; and you can freely combine rules to create complex policies.

Tokens have a set of public and protected actions that you can use to manage the token. Public actions are available to everyone and don't require any authorization. They have similar APIs to coins, but operate on the Token type:

See Coin Token Comparison for coin and token methods comparison.

Protected actions are ones that issue an ActionRequest - a hot-potato struct that must be resolved for the transaction to succeed. There are three main ways to resolve an ActionRequest , most common of which is via the TokenPolicy .

The previous methods are included in the base implementation, however it is possible to create ActionRequest s for custom actions.

Protected actions are disabled by default but you can enable them in a TokenPolicy . Additionally, you can set custom restrictions

called rules that a specific action must satisfy for it to succeed.

Public actions

Tokens have a set of public and protected actions that you can use to manage the token. Public actions are available to everyone and don't require any authorization. They have similar APIs to coins, but operate on the Token type:

See Coin Token Comparison for coin and token methods comparison.

Protected actions are ones that issue an ActionRequest - a hot-potato struct that must be resolved for the transaction to succeed. There are three main ways to resolve an ActionRequest , most common of which is via the TokenPolicy .

The previous methods are included in the base implementation, however it is possible to create ActionRequest s for custom actions.

Protected actions are disabled by default but you can enable them in a TokenPolicy . Additionally, you can set custom restrictions called rules that a specific action must satisfy for it to succeed.

Protected actions

Protected actions are ones that issue an ActionRequest - a hot-potato struct that must be resolved for the transaction to succeed. There are three main ways to resolve an ActionRequest , most common of which is via the TokenPolicy .

The previous methods are included in the base implementation, however it is possible to create ActionRequest s for custom actions.

Protected actions are disabled by default but you can enable them in a TokenPolicy . Additionally, you can set custom restrictions called rules that a specific action must satisfy for it to succeed.

Token policy and rules

Protected actions are disabled by default but you can enable them in a TokenPolicy . Additionally, you can set custom restrictions called rules that a specific action must satisfy for it to succeed.