Token Policy

TokenPolicy is a shared object that you, as the token owner, can create using the TreasuryCap . Having a publicly available TokenPolicy enables on-chain discovery of allowed actions and their conditions. This is useful for wallets and other services that want to provide a better user experience for token holders.

You create a new TokenPolicy using the token::new_policy function. The function takes the TreasuryCap as an argument and returns a TokenPolicy object and a managing capability.

You must use the token::share_policy function to share the TokenPolicy object.

To allow methods without any conditions, use the token::allow function. The function takes a TokenPolicy and TokenPolicyCap as arguments. If allowed, the action can be confirmed in the TokenPolicy using the token::confirm_request function (see ActionRequest).

Similarly, you can use the token::disallow function to completely disable an action; it takes the same arguments as token::allow .

TokenPolicy can specify custom conditions for each action. These conditions are called rules and are typically implemented as separate Move modules. The identifier of the rule is its type. See Rules for more information.

The pseudo-code structure of the TokenPolicy is as follows. Each action can have multiple rules associated with it.

To add a rule for an action, use the token::add_rule_for_action function. The function takes a TokenPolicy and TokenPolicyCap as arguments. The rule is specified by its type (for example, 0x0...::denylist::Denylist).

Signature for the reverse operation token::remove_rule_for_action is symmetrical to token::add_rule_for_action .

Spent balance can be consumed from the TokenPolicy using the token::flush function. It requires a TreasuryCap .

Create and share

You create a new TokenPolicy using the token::new_policy function. The function takes the TreasuryCap as an argument and returns a TokenPolicy object and a managing capability.

You must use the token::share_policy function to share the TokenPolicy object.

To allow methods without any conditions, use the token::allow function. The function takes a TokenPolicy and TokenPolicyCap as arguments. If allowed, the action can be confirmed in the TokenPolicy using the token::confirm_request function (see ActionRequest).

Similarly, you can use the token::disallow function to completely disable an action; it takes the same arguments as token::allow .

TokenPolicy can specify custom conditions for each action. These conditions are called rules and are typically implemented as separate Move modules. The identifier of the rule is its type. See Rules for more information.

The pseudo-code structure of the TokenPolicy is as follows. Each action can have multiple rules associated with it.

To add a rule for an action, use the token::add_rule_for_action function. The function takes a TokenPolicy and TokenPolicyCap as arguments. The rule is specified by its type (for example, 0x0...::denylist::Denylist).

Signature for the reverse operation token::remove_rule_for_action is symmetrical to token::add_rule_for_action .

Spent balance can be consumed from the TokenPolicy using the token::flush function. It requires a TreasuryCap .

Allow and disallow

To allow methods without any conditions, use the token::allow function. The function takes a TokenPolicy and TokenPolicyCap as arguments. If allowed, the action can be confirmed in the TokenPolicy using the token::confirm_request function (see ActionRequest).

Similarly, you can use the token::disallow function to completely disable an action; it takes the same arguments as token::allow .

TokenPolicy can specify custom conditions for each action. These conditions are called rules and are typically implemented as

separate Move modules. The identifier of the rule is its type. See Rules for more information.

The pseudo-code structure of the TokenPolicy is as follows. Each action can have multiple rules associated with it.

To add a rule for an action, use the token::add_rule_for_action function. The function takes a TokenPolicy and TokenPolicyCap as arguments. The rule is specified by its type (for example, 0x0...::denylist::Denylist).

Signature for the reverse operation token::remove_rule_for_action is symmetrical to token::add_rule_for_action .

Spent balance can be consumed from the TokenPolicy using the token::flush function. It requires a TreasuryCap .

Adding rules

TokenPolicy can specify custom conditions for each action. These conditions are called rules and are typically implemented as separate Move modules. The identifier of the rule is its type. See Rules for more information.

The pseudo-code structure of the TokenPolicy is as follows. Each action can have multiple rules associated with it.

To add a rule for an action, use the token::add_rule_for_action function. The function takes a TokenPolicy and TokenPolicyCap as arguments. The rule is specified by its type (for example, 0x0...::denylist::Denylist).

Signature for the reverse operation token::remove_rule_for_action is symmetrical to token::add_rule_for_action .

Spent balance can be consumed from the TokenPolicy using the token::flush function. It requires a TreasuryCap .

Consume spent balance

Spent balance can be consumed from the TokenPolicy using the token::flush function. It requires a TreasuryCap .

Cheatsheet: TokenPolicy API