Regulated Coin and Deny List

You can create regulated coins on Sui, such as stablecoins . These coins are similar to other coins, like SUI, but include the ability to control access to the coin using a deny list.

When creating standard coins, you call the create_currency function in the coin package of the Sui framework, whether directly or via an SDK. When you create regulated coins, you call the create_regulated_currency_v2 function in that same package instead. The create_regulated_currency_v2 function actually leverages the create_currency function to create the coin, but adds an additional step that produces a DenyCapV2 and transfers it to the publisher of the regulated coin package. The bearer of the transferrable DenyCapV2 object can control access to the coin through a deny list.

The DenyList is a singleton, shared object that the bearer of a DenyCapV2 can access to specify a list of addresses that are unable to use a Sui core type. The initial use case for DenyList , however, focuses on limiting access to coins of a specified type. This is useful when creating a regulated coin on Sui that requires the ability to block certain addresses from using it as inputs to transactions. Regulated coins on Sui satisfy any regulations that require the ability to prevent known bad actors from having access to those coins.

The DenyList object is a system object that has the address 0x403 . You cannot create it yourself.

To learn about the features available, see the Coin standard documentation and the coin module in the Sui framework .

The regulated coin example is in the examples/regulated-coin directory of the Sui repo. The example provides both TypeScript- and Rust-based command line access to an on-chain package that demonstrates some of the features of regulated coins on Sui.

This topic assumes you are accessing the code from your own fork of the Sui repo. To run the example project, you must have Sui installed .

You need at least one Sui address to publish the contract to the network. At least one additional address is helpful if you want to transfer or test the deny list capability for the regulated coins.

You do not need a Sui wallet to use this project, but having one available might help you visualize results.

This example assumes you're familiar with publishing packages on Sui and the Move language. For more detailed guides on example dApps, see App Examples . For more more information on the Move language, see The Move Book .

You publish the smart contract to a network the same way as any other package. See Publish a Package if you would like more details on the publishing process.

The example includes a publish.sh file that you can run to automate the publishing. The script assumes you are publishing to the Testnet network, so be sure to update it before running if you plan to run on a local network or Devnet.

The publish script also creates the necessary .env files in each of the frontend folders. If you don't use the script, you must create the .env file manually and provide the values for the variables the frontend expects to find. Even if you use the script, you must provide the ADMIN_SECRET_KEY and it's value.

Take care not to expose the secret key for your address to the public.

The example uses a single file to create the smart contract for the project (regulated_coin.move). The contract defines the regulated coin when you publish it to the network. The treasury capability (TreasuryCap) and deny capability (DenyCapV2) are transferred to the address that publishes the contract. The TreasuryCap permits the bearer to mint or burn coins (REGULATED_COIN in this example), and the DenyCapV2 bearer can add and remove addresses from the list of unauthorized users.

regulated_coin.move

The Sui Coin standard provides a create_regulated_currency_v2 function to create regulated coins. This function actually uses create_currency to mint a coin, but extends the function by also creating and transferring a DenyCapV2 capability. The DenyCapV2 bearer can add and remove addresses from a list that controls, or regulates, access to the coin. This ability is a requirement for assets like stablecoins.

The TypeScript and Rust clients handle the call to the coin package's mint function. The coin package also includes a mint_and_transfer function you could use to perform the same task, but the composability of minting the coin in one command and transferring with another is preferable. Using two explicit commands allows you to implement future logic between the minting of the coin and the transfer. The structure of programmable transaction blocks means you're still making and paying for a single transaction on the network.

For all Coin functions available, see the Sui framework coin module documentation . The following functions are the most common.

coin::mint

coin::mint_balance

coin::mint_and_transfer

coin::burn

For the ability to manage the addresses assigned to the deny list for your coin, the frontend code provides a few additional functions. These additions call the deny_list_v2_add and deny_list_v2_remove functions in the coin module.

If you add an address to the deny list, you might notice that you can still send tokens to that address. If so, that's because the address is still able to receive coins until the end of the epoch in which you called the function. If you try to send the regulated coin from the now blocked address, your attempt results in an error. After the next epoch starts, the address can no longer receive the coins, either. If you remove the address, it can receive coins immediately but must wait until the epoch after removal before the address can include the coins as transaction inputs.

To use these functions, you pass the address you want to either add or remove. The frontend function then calls the relevant move module in the framework, adding the DenyList object (0x403) and your DenyCap object ID. You receive the DenyCap ID at the time of publishing the smart contract. In this example, you add that value to the .env file that the frontend function reads from.

DenyList

The DenyList is a singleton, shared object that the bearer of a DenyCapV2 can access to specify a list of addresses that are unable to use a Sui core type. The initial use case for DenyList , however, focuses on limiting access to coins of a specified type. This is useful when creating a regulated coin on Sui that requires the ability to block certain addresses from using it as inputs to transactions. Regulated coins on Sui satisfy any regulations that require the ability to prevent known bad actors from having access to those coins.

The DenyList object is a system object that has the address 0x403 . You cannot create it yourself.

To learn about the features available, see the Coin standard documentation and the coin module in the Sui framework .

The regulated coin example is in the examples/regulated-coin directory of the Sui repo. The example provides both TypeScript- and Rust-based command line access to an on-chain package that demonstrates some of the features of regulated coins on Sui.

This topic assumes you are accessing the code from your own fork of the Sui repo. To run the example project, you must have Sui installed .

You need at least one Sui address to publish the contract to the network. At least one additional address is helpful if you want to transfer or test the deny list capability for the regulated coins.

You do not need a Sui wallet to use this project, but having one available might help you visualize results.

This example assumes you're familiar with publishing packages on Sui and the Move language. For more detailed guides on example dApps, see App Examples . For more more information on the Move language, see The Move Book .

You publish the smart contract to a network the same way as any other package. See Publish a Package if you would like more details on the publishing process.

The example includes a publish.sh file that you can run to automate the publishing. The script assumes you are publishing to the Testnet network, so be sure to update it before running if you plan to run on a local network or Devnet.

The publish script also creates the necessary .env files in each of the frontend folders. If you don't use the script, you must create the .env file manually and provide the values for the variables the frontend expects to find. Even if you use the script, you must provide the ADMIN_SECRET_KEY and it's value.

Take care not to expose the secret key for your address to the public.

The example uses a single file to create the smart contract for the project (regulated_coin.move). The contract defines the regulated coin when you publish it to the network. The treasury capability (TreasuryCap) and deny capability (DenyCapV2) are transferred to the address that publishes the contract. The TreasuryCap permits the bearer to mint or burn coins (REGULATED_COIN in this example), and the DenyCapV2 bearer can add and remove addresses from the list of unauthorized users.

regulated_coin.move

The Sui Coin standard provides a create_regulated_currency_v2 function to create regulated coins. This function actually uses create_currency to mint a coin, but extends the function by also creating and transferring a DenyCapV2 capability. The DenyCapV2 bearer can add and remove addresses from a list that controls, or regulates, access to the coin. This ability is a requirement for assets like stablecoins.

The TypeScript and Rust clients handle the call to the coin package's mint function. The coin package also includes a mint_and_transfer function you could use to perform the same task, but the composability of minting the coin in one command and transferring with another is preferable. Using two explicit commands allows you to implement future logic between the minting of the coin and the transfer. The structure of programmable transaction blocks means you're still making and paying for a single transaction on the network.

For all Coin functions available, see the Sui framework coin module documentation . The following functions are the most common.

coin::mint

coin::mint_balance

coin::mint_and_transfer

coin::burn

For the ability to manage the addresses assigned to the deny list for your coin, the frontend code provides a few additional functions. These additions call the deny_list_v2_add and deny_list_v2_remove functions in the coin module.

If you add an address to the deny list, you might notice that you can still send tokens to that address. If so, that's because the address is still able to receive coins until the end of the epoch in which you called the function. If you try to send the regulated coin from the now blocked address, your attempt results in an error. After the next epoch starts, the address can no longer receive the coins, either. If you remove the address, it can receive coins immediately but must wait until the epoch after removal before the address can include the coins as transaction inputs.

To use these functions, you pass the address you want to either add or remove. The frontend function then calls the relevant move module in the framework, adding the DenyList object (0x403) and your DenyCap object ID. You receive the DenyCap ID at the time of publishing the smart contract. In this example, you add that value to the .env file that the frontend function reads from.

Regulated coin example

The regulated coin example is in the examples/regulated-coin directory of the Sui repo. The example provides both TypeScript- and Rust-based command line access to an on-chain package that demonstrates some of the features of regulated coins on Sui.

This topic assumes you are accessing the code from your own fork of the Sui repo. To run the example project, you must have Sui installed .

You need at least one Sui address to publish the contract to the network. At least one additional address is helpful if you want to transfer or test the deny list capability for the regulated coins.

You do not need a Sui wallet to use this project, but having one available might help you visualize results.

This example assumes you're familiar with publishing packages on Sui and the Move language. For more detailed guides on example dApps, see App Examples . For more more information on the Move language, see The Move Book .

You publish the smart contract to a network the same way as any other package. See Publish a Package if you would like more details on the publishing process.

The example includes a publish.sh file that you can run to automate the publishing. The script assumes you are publishing to the Testnet network, so be sure to update it before running if you plan to run on a local network or Devnet.

The publish script also creates the necessary .env files in each of the frontend folders. If you don't use the script, you must create the .env file manually and provide the values for the variables the frontend expects to find. Even if you use the script, you must provide the ADMIN_SECRET_KEY and it's value.

Take care not to expose the secret key for your address to the public.

The example uses a single file to create the smart contract for the project (regulated_coin.move). The contract defines the regulated coin when you publish it to the network. The treasury capability (TreasuryCap) and deny capability (DenyCapV2) are transferred to the address that publishes the contract. The TreasuryCap permits the bearer to mint or burn coins (REGULATED_COIN in this example), and the DenyCapV2 bearer can add and remove addresses from the list of unauthorized users.

regulated_coin.move

The Sui Coin standard provides a create_regulated_currency_v2 function to create regulated coins. This function actually uses create_currency to mint a coin, but extends the function by also creating and transferring a DenyCapV2 capability. The DenyCapV2 bearer can add and remove addresses from a list that controls, or regulates, access to the coin. This ability is a requirement for assets like stablecoins.

The TypeScript and Rust clients handle the call to the coin package's mint function. The coin package also includes a mint_and_transfer function you could use to perform the same task, but the composability of minting the coin in one command and transferring with another is preferable. Using two explicit commands allows you to implement future logic between the minting of the coin and the transfer. The structure of programmable transaction blocks means you're still making and paying for a single transaction on the network.

For all Coin functions available, see the Sui framework coin module documentation . The following functions are the most common.

coin::mint

coin::mint_balance

coin::mint_and_transfer

coin::burn

For the ability to manage the addresses assigned to the deny list for your coin, the frontend code provides a few additional functions. These additions call the deny_list_v2_add and deny_list_v2_remove functions in the coin module.

If you add an address to the deny list, you might notice that you can still send tokens to that address. If so, that's because the address is still able to receive coins until the end of the epoch in which you called the function. If you try to send the regulated coin from the now blocked address, your attempt results in an error. After the next epoch starts, the address can no longer receive the coins, either. If you remove the address, it can receive coins immediately but must wait until the epoch after removal before the address can include the coins as transaction inputs.

To use these functions, you pass the address you want to either add or remove. The frontend function then calls the relevant move module in the framework, adding the DenyList object (0x403) and your DenyCap object ID. You receive the DenyCap ID at the time of publishing the smart contract. In this example, you add that value to the .env file that the frontend function reads from.

Prerequisites

This topic assumes you are accessing the code from your own fork of the Sui repo. To run the example project, you must have Sui installed .

You need at least one Sui address to publish the contract to the network. At least one additional address is helpful if you want to transfer or test the deny list capability for the regulated coins.

You do not need a Sui wallet to use this project, but having one available might help you visualize results.

This example assumes you're familiar with publishing packages on Sui and the Move language. For more detailed guides on example dApps, see App Examples . For more more information on the Move language, see The Move Book .

You publish the smart contract to a network the same way as any other package. See Publish a Package if you would like more details on the publishing process.

The example includes a publish.sh file that you can run to automate the publishing. The script assumes you are publishing to the Testnet network, so be sure to update it before running if you plan to run on a local network or Devnet.

The publish script also creates the necessary .env files in each of the frontend folders. If you don't use the script, you must create the .env file manually and provide the values for the variables the frontend expects to find. Even if you use the script, you must provide the ADMIN_SECRET_KEY and it's value.

Take care not to expose the secret key for your address to the public.

The example uses a single file to create the smart contract for the project (regulated_coin.move). The contract defines the regulated coin when you publish it to the network. The treasury capability (TreasuryCap) and deny capability (DenyCapV2) are transferred to the address that publishes the contract. The TreasuryCap permits the bearer to mint or burn coins (REGULATED_COIN in this example), and the DenyCapV2 bearer can add and remove addresses from the list of unauthorized users.

regulated_coin.move

The Sui Coin standard provides a create_regulated_currency_v2 function to create regulated coins. This function actually uses create_currency to mint a coin, but extends the function by also creating and transferring a DenyCapV2 capability. The DenyCapV2 bearer can add and remove addresses from a list that controls, or regulates, access to the coin. This ability is a requirement for assets like stablecoins.

The TypeScript and Rust clients handle the call to the coin package's mint function. The coin package also includes a mint_and_transfer function you could use to perform the same task, but the composability of minting the coin in one command and transferring with another is preferable. Using two explicit commands allows you to implement future logic between the minting of the coin and the transfer. The structure of programmable transaction blocks means you're still making and paying for a single transaction on the network.

For all Coin functions available, see the Sui framework coin module documentation . The following functions are the most common.

coin::mint

coin::mint_balance

coin::mint_and_transfer

coin::burn

For the ability to manage the addresses assigned to the deny list for your coin, the frontend code provides a few additional functions. These additions call the deny_list_v2_add and deny_list_v2_remove functions in the coin module.

If you add an address to the deny list, you might notice that you can still send tokens to that address. If so, that's because the address is still able to receive coins until the end of the epoch in which you called the function. If you try to send the regulated coin from the now blocked address, your attempt results in an error. After the next epoch starts, the address can no longer receive the coins, either. If you remove the address, it can receive coins immediately but must wait until the epoch after removal before the address can include the coins as transaction inputs.

To use these functions, you pass the address you want to either add or remove. The frontend function then calls the relevant move module in the framework, adding the DenyList object (0x403) and your DenyCap object ID. You receive the DenyCap ID at the time of publishing the smart contract. In this example, you add that value to the .env file that the frontend function reads from.

Publishing to a network

You publish the smart contract to a network the same way as any other package. See Publish a Package if you would like more details on the publishing process.

The example includes a publish.sh file that you can run to automate the publishing. The script assumes you are publishing to the Testnet network, so be sure to update it before running if you plan to run on a local network or Devnet.

The publish script also creates the necessary .env files in each of the frontend folders. If you don't use the script, you must create the .env file manually and provide the values for the variables the frontend expects to find. Even if you use the script, you must provide the ADMIN_SECRET_KEY and it's value.

Take care not to expose the secret key for your address to the public.

The example uses a single file to create the smart contract for the project (regulated_coin.move). The contract defines the regulated coin when you publish it to the network. The treasury capability (TreasuryCap) and deny capability (DenyCapV2) are transferred to the address that publishes the contract. The TreasuryCap permits the bearer to mint or burn coins (REGULATED_COIN in this example), and the DenyCapV2 bearer can add and remove addresses from the list of unauthorized users.

regulated_coin.move

The Sui Coin standard provides a create_regulated_currency_v2 function to create regulated coins. This function actually uses create_currency to mint a coin, but extends the function by also creating and transferring a DenyCapV2 capability. The DenyCapV2 bearer can add and remove addresses from a list that controls, or regulates, access to the coin. This ability is a requirement for assets like stablecoins.

The TypeScript and Rust clients handle the call to the coin package's mint function. The coin package also includes a mint_and_transfer function you could use to perform the same task, but the composability of minting the coin in one command and transferring with another is preferable. Using two explicit commands allows you to implement future logic between the minting of the coin and the transfer. The structure of programmable transaction blocks means you're still making and paying for a single transaction on the network.

For all Coin functions available, see the Sui framework coin module documentation . The following functions are the most common.

coin::mint

coin::mint_balance

coin::mint_and_transfer

coin::burn

For the ability to manage the addresses assigned to the deny list for your coin, the frontend code provides a few additional functions. These additions call the deny_list_v2_add and deny_list_v2_remove functions in the coin module.

If you add an address to the deny list, you might notice that you can still send tokens to that address. If so, that's because the address is still able to receive coins until the end of the epoch in which you called the function. If you try to send the regulated coin from the now blocked address, your attempt results in an error. After the next epoch starts, the address can no longer receive the coins, either. If you remove the address, it can receive coins immediately but must wait until the epoch after removal before the address can include the coins as transaction inputs.

To use these functions, you pass the address you want to either add or remove. The frontend function then calls the relevant move module in the framework, adding the DenyList object (0x403) and your DenyCap object ID. You receive the DenyCap ID at the time of publishing the smart contract. In this example, you add that value to the .env file that the frontend function reads from.

Smart contract

The example uses a single file to create the smart contract for the project (regulated_coin.move). The contract defines the regulated coin when you publish it to the network. The treasury capability (TreasuryCap) and deny capability (DenyCapV2) are transferred to the address that publishes the contract. The TreasuryCap permits the bearer to mint or burn coins (REGULATED_COIN in this example), and the DenyCapV2 bearer can add and remove addresses from the list of unauthorized users.

regulated_coin.move

The Sui Coin standard provides a create_regulated_currency_v2 function to create regulated coins. This function actually uses create_currency to mint a coin, but extends the function by also creating and transferring a DenyCapV2 capability. The DenyCapV2 bearer can add and remove addresses from a list that controls, or regulates, access to the coin. This ability is a requirement for assets like stablecoins.

The TypeScript and Rust clients handle the call to the coin package's mint function. The coin package also includes a mint_and_transfer function you could use to perform the same task, but the composability of minting the coin in one command and transferring with another is preferable. Using two explicit commands allows you to implement future logic between the minting of the coin and the transfer. The structure of programmable transaction blocks means you're still making and paying for a single transaction on the network.

For all Coin functions available, see the Sui framework coin module documentation . The following functions are the most common.

coin::mint

coin::mint_balance

coin::mint_and_transfer

coin::burn

For the ability to manage the addresses assigned to the deny list for your coin, the frontend code provides a few additional functions. These additions call the deny_list_v2_add and deny_list_v2_remove functions in the coin module.

If you add an address to the deny list, you might notice that you can still send tokens to that address. If so, that's because the address is still able to receive coins until the end of the epoch in which you called the function. If you try to send the regulated coin from the now blocked address, your attempt results in an error. After the next epoch starts, the address can no longer receive the coins, either. If you remove the address, it can receive coins immediately but must wait until the epoch after removal before the address can include the coins as transaction inputs.

To use these functions, you pass the address you want to either add or remove. The frontend function then calls the relevant move module in the framework, adding the DenyList object (0x403) and your DenyCap object ID. You receive the DenyCap ID at the time of publishing the smart contract. In this example, you add that value to the .env file that the frontend function reads from.

Related links