Building Programmable Transaction Blocks

This guide explores creating a programmable transaction block (PTB) on Sui using the TypeScript SDK. For an overview of what a PTB is, see [Programmable Transaction Blocks]() in the Concepts section. If you don't already have the Sui TypeScript SDK, follow the [install instructions]() on the Sui TypeScript SDK site.

This example starts by constructing a PTB to send Sui. If you are familiar with the legacy Sui transaction types, this is similar to a paySui transaction. To construct transactions, import the Transaction class, and construct it:

Using this, you can then add transactions to this PTB.

You can attach multiple transaction commands of the same type to a PTB as well. For example, to get a list of transfers, and iterate over them to transfer coins to each of them:

After you have the Transaction defined, you can directly execute it with a SuiClient and KeyPair using client.signAndExecuteTransaction .

Inputs are how you provide external values to PTBs. For example, defining an amount of Sui to transfer, or which object to pass into a Move call, or a shared object.

There are currently two ways to define inputs:

Sui supports following transaction commands:

You can use the result of a transaction command as an argument in subsequent transaction commands. Each transaction command method on the transaction builder returns a reference to the transaction result.

When a transaction command returns multiple results, you can access the result at a specific index either using destructuring, or array indexes.

With PTBs, you can use the gas payment coin to construct coins with a set balance using splitCoin . This is useful for Sui payments, and avoids the need for up-front coin selection. You can use tx.gas to access the gas coin in a PTB, and it is valid as input for any arguments; with the exception of transferObjects , tx.gas must be used by-reference. Practically speaking, this means you can also add to the gas coin with mergeCoins or borrow it for Move functions with moveCall .

You can also transfer the gas coin using transferObjects , in the event that you want to transfer all of your coin balance to another address.

Of course, you can also transfer other coins in your wallet using their Object ID . For example,

If you need the PTB bytes, instead of signing or executing the PTB, you can use the build method on the transaction builder itself.

You might need to explicitly call setSender() on the PTB to ensure that the sender field is populated. This is normally done by the signer before signing the transaction, but will not be done automatically if you're building the PTB bytes yourself.

In most cases, building requires your JSON RPC provider to fully resolve input values.

If you have PTB bytes, you can also convert them back into a Transaction class:

In the event that you want to build a PTB offline (as in with no provider required), you need to fully define all of your input values, and gas configuration (see the following example). For pure values, you can provide a Uint8Array which is used directly in the transaction. For objects, you can use the Inputs helper to construct an object reference.

You can then omit the provider object when calling build on the transaction. If there is any required data that is missing, this will throw an error.

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Constructing inputs

Inputs are how you provide external values to PTBs. For example, defining an amount of Sui to transfer, or which object to pass into a Move call, or a shared object.

There are currently two ways to define inputs:

Sui supports following transaction commands:

You can use the result of a transaction command as an argument in subsequent transaction commands. Each transaction command method on the transaction builder returns a reference to the transaction result.

When a transaction command returns multiple results, you can access the result at a specific index either using destructuring, or array indexes.

With PTBs, you can use the gas payment coin to construct coins with a set balance using splitCoin . This is useful for Sui payments, and avoids the need for up-front coin selection. You can use tx.gas to access the gas coin in a PTB, and it is valid as input for any arguments; with the exception of transferObjects , tx.gas must be used by-reference. Practically speaking, this means you can also add to the gas coin with mergeCoins or borrow it for Move functions with moveCall .

You can also transfer the gas coin using transferObjects , in the event that you want to transfer all of your coin balance to another address.

Of course, you can also transfer other coins in your wallet using their Object ID . For example,

If you need the PTB bytes, instead of signing or executing the PTB, you can use the build method on the transaction builder itself.

You might need to explicitly call setSender() on the PTB to ensure that the sender field is populated. This is normally done by the signer before signing the transaction, but will not be done automatically if you're building the PTB bytes yourself.

In most cases, building requires your JSON RPC provider to fully resolve input values.

If you have PTB bytes, you can also convert them back into a Transaction class:

In the event that you want to build a PTB offline (as in with no provider required), you need to fully define all of your input values, and gas configuration (see the following example). For pure values, you can provide a Uint8Array which is used directly in the transaction. For objects, you can use the Inputs helper to construct an object reference.

You can then omit the provider object when calling build on the transaction. If there is any required data that is missing, this will throw an error.

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Available transactions

Sui supports following transaction commands:

You can use the result of a transaction command as an argument in subsequent transaction commands. Each transaction command method on the transaction builder returns a reference to the transaction result.

When a transaction command returns multiple results, you can access the result at a specific index either using destructuring, or array indexes.

With PTBs, you can use the gas payment coin to construct coins with a set balance using splitCoin . This is useful for Sui payments, and avoids the need for up-front coin selection. You can use tx.gas to access the gas coin in a PTB, and it is valid as input for any arguments; with the exception of transferObjects , tx.gas must be used by-reference. Practically speaking, this means you can also add to the gas coin with mergeCoins or borrow it for Move functions with moveCall .

You can also transfer the gas coin using transferObjects , in the event that you want to transfer all of your coin balance to another address.

Of course, you can also transfer other coins in your wallet using their Object ID . For example,

If you need the PTB bytes, instead of signing or executing the PTB, you can use the build method on the transaction builder itself.

You might need to explicitly call setSender() on the PTB to ensure that the sender field is populated. This is normally done by the signer before signing the transaction, but will not be done automatically if you're building the PTB bytes yourself.

In most cases, building requires your JSON RPC provider to fully resolve input values.

If you have PTB bytes, you can also convert them back into a Transaction class:

In the event that you want to build a PTB offline (as in with no provider required), you need to fully define all of your input values, and gas configuration (see the following example). For pure values, you can provide a Uint8Array which is used directly in the transaction. For objects, you can use the Inputs helper to construct an object reference.

You can then omit the provider object when calling build on the transaction. If there is any required data that is missing, this will throw an error.

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Passing transaction results as arguments

You can use the result of a transaction command as an argument in subsequent transaction commands. Each transaction command method on the transaction builder returns a reference to the transaction result.

When a transaction command returns multiple results, you can access the result at a specific index either using destructuring, or array indexes.

With PTBs, you can use the gas payment coin to construct coins with a set balance using splitCoin . This is useful for Sui payments, and avoids the need for up-front coin selection. You can use tx.gas to access the gas coin in a PTB, and it is valid as input for any arguments; with the exception of transferObjects , tx.gas must be used by-reference. Practically speaking, this means you can also add to the gas coin with mergeCoins or borrow it for Move functions with moveCall .

You can also transfer the gas coin using transferObjects , in the event that you want to transfer all of your coin balance to another address.

Of course, you can also transfer other coins in your wallet using their Object ID . For example,

If you need the PTB bytes, instead of signing or executing the PTB, you can use the build method on the transaction builder itself.

You might need to explicitly call setSender() on the PTB to ensure that the sender field is populated. This is normally done by the signer before signing the transaction, but will not be done automatically if you're building the PTB bytes yourself.

In most cases, building requires your JSON RPC provider to fully resolve input values.

If you have PTB bytes, you can also convert them back into a Transaction class:

In the event that you want to build a PTB offline (as in with no provider required), you need to fully define all of your input values, and gas configuration (see the following example). For pure values, you can provide a Uint8Array which is used directly in the transaction. For objects, you can use the Inputs helper to construct an object reference.

You can then omit the provider object when calling build on the transaction. If there is any required data that is missing, this will throw an error.

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Use the gas coin

With PTBs, you can use the gas payment coin to construct coins with a set balance using splitCoin . This is useful for Sui payments, and avoids the need for up-front coin selection. You can use tx.gas to access the gas coin in a PTB, and it is valid as input for any arguments; with the exception of transferObjects , tx.gas must be used by-reference. Practically speaking, this means you can also add to the gas coin with mergeCoins or borrow it for Move functions with moveCall .

You can also transfer the gas coin using transferObjects , in the event that you want to transfer all of your coin balance to another address.

Of course, you can also transfer other coins in your wallet using their Object ID . For example,

If you need the PTB bytes, instead of signing or executing the PTB, you can use the build method on the transaction builder itself.

You might need to explicitly call setSender() on the PTB to ensure that the sender field is populated. This is normally done by the signer before signing the transaction, but will not be done automatically if you're building the PTB bytes yourself.

In most cases, building requires your JSON RPC provider to fully resolve input values.

If you have PTB bytes, you can also convert them back into a Transaction class:

In the event that you want to build a PTB offline (as in with no provider required), you need to fully define all of your input values, and gas configuration (see the following example). For pure values, you can provide a Uint8Array which is used directly in the transaction. For objects, you can use the Inputs helper to construct an object reference.

You can then omit the provider object when calling build on the transaction. If there is any required data that is missing, this will throw an error.

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Get PTB bytes

If you need the PTB bytes, instead of signing or executing the PTB, you can use the build method on the transaction builder itself.

You might need to explicitly call setSender() on the PTB to ensure that the sender field is populated. This is normally done by the signer before signing the transaction, but will not be done automatically if you're building the PTB bytes yourself.

In most cases, building requires your JSON RPC provider to fully resolve input values.

If you have PTB bytes, you can also convert them back into a Transaction class:

In the event that you want to build a PTB offline (as in with no provider required), you need to fully define all of your input values, and gas configuration (see the following example). For pure values, you can provide a Uint8Array which is used directly in the transaction. For objects, you can use the Inputs helper to construct an object reference.

You can then omit the provider object when calling build on the transaction. If there is any required data that is missing, this will throw an error.

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Building offline

In the event that you want to build a PTB offline (as in with no provider required), you need to fully define all of your input values, and gas configuration (see the following example). For pure values, you can provide a Uint8Array which is used directly in the transaction. For objects, you can use the Inputs helper to construct an object reference.

You can then omit the provider object when calling build on the transaction. If there is any required data that is missing, this will throw an error.

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Gas configuration

The new transaction builder comes with default behavior for all gas logic, including automatically setting the gas price, budget, and selecting coins to be used as gas. This behavior can be customized.

By default, the gas price is set to the reference gas price of the network. You can also explicitly set the gas price of the PTB by calling setGasPrice on the transaction builder.

By default, the gas budget is automatically derived by executing a dry-run of the PTB beforehand. The dry run gas consumption is then used to determine a balance for the transaction. You can override this behavior by explicitly setting a gas budget for the transaction, by calling setGasBudget on the transaction builder.

The gas budget is represented in Sui, and should take the gas price of the PTB into account.

By default, the gas payment is automatically determined by the SDK. The SDK selects all coins at the provided address that are not used as inputs in the PTB.

The list of coins used as gas payment will be merged down into a single gas coin before executing the PTB, and all but one of the gas objects will be deleted. The gas coin at the 0-index will be the coin that all others are merged into.

Gas coins should be objects containing the coins objectId, version, and digest (ie { objectId: string, version: string | number, digest: string }).

The Wallet Standard interface has been updated to support the Transaction kind directly. All signTransaction and signAndExecuteTransaction calls from dApps into wallets is expected to provide a Transaction class. This PTB class can then be serialized and sent to your wallet for execution.

To serialize a PTB for sending to a wallet, Sui recommends using the tx.serialize() function, which returns an opaque string representation of the PTB that can be passed from the wallet standard dApp context to your wallet. This can then be converted back into a Transaction using Transaction.from() .

You should not build the PTB from bytes in the dApp code. Using serialize instead of build allows you to build the PTB bytes within the wallet itself. This allows the wallet to perform gas logic and coin selection as needed.

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Sponsored PTBs

The PTB builder can support sponsored PTBs by using the onlyTransactionKind flag when building the PTB.

Related links