

# Safe reference

## Initialization

### connect

Returns a new instance of the Protocol Kit connected to a new Safe or a new Signer. The new connected signer can be passed via the `ethAdapter` property while the new connected Safe can be passed using `safeAddress` or `apredictedSafe`.

Connection of a deployed Safe using the `safeAddress` property:

```
let protocolKit =  
await  
Safe.create ({ ethAdapter , safeAddress })  
await  
protocolKit.connect ({ ethAdapter : anotherEthAdapter , safeAddress : anotherSafeAddress })
```

Connection of an undeployed Safe using the `apredictedSafe` property. Because Safes are deployed in a deterministic way, passing `apredictedSafe` will allow to connect a Safe to the SDK with the Safe configuration:

```
import { PredictedSafeProps } from  
'@safe-global/protocol-kit'  
const  
predictedSafe :  
PredictedSafeProps  
= { safeAccountConfig , safeDeploymentConfig }  
let protocolKit =  
await  
Safe.create ({ ethAdapter , safeAddress }) ... protocolKit =  
await  
protocolKit.connect ({ predictedSafe })
```

- `TheisL1SafeSingleton`
- `flag`
- Two versions of the Safe contracts are available [Safe.sol\(opens in a new tab\)](#)
- that doesn't trigger events to save gas and [SafeL2.sol\(opens in a new tab\)](#)
- that does, which is more appropriate for L2 networks.
- By default `Safe.sol`
- will only be used on Ethereum Mainnet. For the rest of the networks where the Safe contracts are already deployed, `theSafeL2.sol`
- contract will be used unless you add `theisL1SafeSingleton`
- `flag` to force using `theSafe.sol`
- `contract`.
- `protocolKit`
- `=`
- `await`
- `protocolKit`
- `.connect`
- `( { ethAdapter`
- `,`
- `safeAddress`
- `,`
- `isL1SafeSingleton`
- `:`
- `true`
- `})`

- ThecontractNetworks
- property
- If the Safe contracts aren't deployed to your current network, thecontractNetworks
- property will be required to point to the addresses of the Safe contracts previously deployed by you.
- import
- { ContractNetworksConfig }
- from
- '@safe-global/protocol-kit'
- const
- chainId
- =
- await
- ethAdapter
- .getChainId
- ()
- const
- contractNetworks
- :
- ContractNetworksConfig
- =
- {
- [chainId]
- :
- {
- safeSingletonAddress
- :
- "
- ,
- safeProxyFactoryAddress
- :
- "
- ,
- multiSendAddress
- :
- "
- ,
- multiSendCallOnlyAddress
- :
- "
- ,
- fallbackHandlerAddress
- :
- "
- ,
- signMessageLibAddress
- :
- "
- ,
- createCallAddress
- :
- "
- ,
- simulateTxAccessorAddress
- :
- "
- ,
- safeSingletonAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- safeProxyFactoryAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- multiSendAbi
- :

- "
- ,
- // Optional. Only needed with web3.js
- multiSendCallOnlyAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- fallbackHandlerAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- signMessageLibAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- createCallAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- simulateTxAccessorAbi
- :
- "
- // Optional. Only needed with web3.js
- }
- }
- let
- protocolKit
- =
- await
- Safe
- .create
- ({ ethAdapter
- ,
- safeAddress })
- ...
- protocolKit
- =
- await
- protocolKit
- .connect
- ({ contractNetworks })

## create

Returns an instance of the Protocol Kit connected to a Safe. The provided Safe must be a `safeAddress` or a `predictedSafe`.

Initialization of a deployed Safe using the `safeAddress` property:

```
import Safe from
```

```
'@safe-global/protocol-kit'
```

```
const
```

```
protocolKit
```

```
=
```

```
await
```

```
Safe .create ({ ethAdapter , safeAddress })
```

Initialization of an undeployed Safe using the `predictedSafe` property. Because Safes are deployed in a deterministic way, passing `predictedSafe` will allow to initialize the SDK with the Safe configuration and use it to some extent before it's deployed:

```
import Safe , { PredictedSafeProps } from
```

```
'@safe-global/protocol-kit'
```

```
const
```

```
predictedSafe :
```

```
PredictedSafeProps
```

```
= { safeAccountConfig , safeDeploymentConfig }
```

```
const
```

```
protocolKit
```

```
=
```

```
await
```

```
Safe .create ({ ethAdapter , predictedSafe })
```

- TheisL1SafeSingleton
- flag
- Two versions of the Safe contracts are available [Safe.sol\(opens in a new tab\)](#)
- that doesn't trigger events to save gas and [SafeL2.sol\(opens in a new tab\)](#)
- that does, which is more appropriate for L2 networks.
- By default Safe.sol
- will only be used on Ethereum Mainnet. For the rest of the networks where the Safe contracts are already deployed, the SafeL2.sol
- contract will be used unless you add theisL1SafeSingleton
- flag to force using the Safe.sol
- contract.
- const
- protocolKit
- =
- await
- Safe
- .create
- ({ ethAdapter
- ,
- safeAddress
- ,
- isL1SafeSingleton
- :
- true
- })
- ThecontractNetworks
- property
- If the Safe contracts aren't deployed to your current network, thecontractNetworks
- property will be required to point to the addresses of the Safe contracts previously deployed by you.
- import
- { ContractNetworksConfig }
- from
- '@safe-global/protocol-kit'
- const
- chainId
- =
- await
- ethAdapter
- .getChainId
- ()
- const
- contractNetworks
- :
- ContractNetworksConfig
- =
- {
- [chainId]
- :

- {
- safeSingletonAddress
- :
- "
- ,
- safeProxyFactoryAddress
- :
- "
- ,
- multiSendAddress
- :
- "
- ,
- multiSendCallOnlyAddress
- :
- "
- ,
- fallbackHandlerAddress
- :
- "
- ,
- signMessageLibAddress
- :
- "
- ,
- createCallAddress
- :
- "
- ,
- simulateTxAccessorAddress
- :
- "
- ,
- safeSingletonAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- safeProxyFactoryAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- multiSendAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- multiSendCallOnlyAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- fallbackHandlerAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- signMessageLibAbi
- :
- "
- ,
- // Optional. Only needed with web3.js
- createCallAbi
- :
- "
- ,
- // Optional. Only needed with web3.js

- simulateTxAccessorAbi
- :
- "
- // Optional. Only needed with web3.js
- }
- }
- const
- protocolKit
- =
- await
- Safe
- .create
- ({ ethAdapter
- ,
- safeAddress
- ,
- contractNetworks })

## Safe Info

### getAddress

Returns the address of the current SafeProxy contract.

```
const
```

```
safeAddress
```

```
=
```

```
await
```

```
protocolKit .getAddress ()
```

### getBalance

Returns the ETH balance of the Safe.

```
const
```

```
balance
```

```
=
```

```
await
```

```
protocolKit .getBalance ()
```

### getChainId

Returns the chain ID of the connected network.

```
const
```

```
chainId
```

```
=
```

```
await
```

```
protocolKit .getChainId ()
```

### getContractVersion

Returns the Safe singleton contract version.

```
const
```

```
contractVersion
```

```
=  
await  
protocolKit .getContractVersion ()
```

## **getNonce**

Returns the Safe nonce.

```
const  
nonce  
=  
await  
protocolKit .getNonce ()
```

## **Transactions**

### **copyTransaction**

Copies a Safe transaction.

```
const  
safeTransaction1  
=  
await  
protocolKit .createTransaction ({ transactions }) const  
safeTransaction2  
=  
await  
copyTransaction (safeTransaction1)
```

### **createRejectionTransaction**

Returns a Safe transaction ready to be signed by the owners that invalidates the pending Safe transaction(s) with a specific nonce.

```
const  
transactions :  
MetaTransactionData [] = [{ // ... }] const  
safeTransaction  
=  
await  
protocolKit .createTransaction ({ transactions }) const  
rejectionTransaction  
=  
await  
protocolKit .createRejectionTransaction ( safeTransaction . data .nonce)
```

### **createTransaction**

Returns a Safe transaction ready to be signed by the owners and executed. The Protocol Kit supports the creation of single Safe transactions but also MultiSend transactions.

This method takes an array of MetaTransactionData objects representing the individual transactions we want to include in our MultiSend transaction.

When the array contains only one transaction, it's not wrapped in the MultiSend.

```
const
transactions :
MetaTransactionData [] = [ { to , data , value , operation // Optional } , { to , data , value , operation // Optional } // ... ] const
safeTransaction
=
await
protocolKit .createTransaction ({ transactions })
```

This method can also receive the options parameter to set the optional properties in the MultiSend transaction:

```
const
transactions :
MetaTransactionData [] = [ { to , data , value , operation // Optional } , { to , data , value , operation // Optional } // ... ] const
options :
SafeTransactionOptionalProps
= { safeTxGas ,
// Optional baseGas ,
// Optional gasPrice ,
// Optional gasToken ,
// Optional refundReceiver ,
// Optional nonce // Optional } const
safeTransaction
=
await
protocolKit .createTransaction ({ transactions , options })
```

In addition, the optional callsOnly parameter, which is false by default, allows forcing the use of the MultiSendCallOnly instead of the MultiSend contract when sending a batch transaction:

```
const
callsOnly
=
true const
safeTransaction
=
await
protocolKit .createTransaction ({ transactions , options , callsOnly })
```

If the optional properties aren't manually set, the Safe transaction returned will have the default value for each one:



- operation
- :OperationType.Call
- (0) is the default value.
- safeTxGas
- : The right gas estimation is the default value.
- baseGas
- : 0 is the default value.
- gasPrice
- : 0 is the default value.
- gasToken
- : 0x address is the default value.
- refundReceiver
- : 0x address is the default value.
- nonce
- : The current Safe nonce is the default value.

## executeTransaction

Executes a Safe transaction.

const

transactions :

MetaTransactionData [] = [{ // ... }] const

safeTransaction

=

await

protocolKit .createTransaction ({ transactions }) const

txResponse

=

await

protocolKit .executeTransaction (safeTransaction) await

txResponse . transactionResponse ?.wait ()

Optionally, some properties can be passed as execution options:

const

options :

Web3TransactionOptions

= { from ,

// Optional gas ,

// Optional gasPrice ,

// Optional maxFeePerGas ,

// Optional maxPriorityFeePerGas // Optional nonce // Optional }

const

options :

EthersTransactionOptions

= { from ,

// Optional gasLimit ,

// Optional gasPrice ,

```
// Optional maxFeePerGas ,
// Optional maxPriorityFeePerGas // Optional nonce // Optional }

const
txResponse
=
await
protocolKit .executeTransaction (safeTransaction , options)
```

## **getTransactionHash**

Returns the transaction hash of a Safe transaction.

```
const
transactions :
MetaTransactionData [] = [{ // ... }] const
safeTransaction
=
await
protocolKit .createTransaction ({ transactions }) const
txHash
=
await
protocolKit .getTransactionHash (safeTransaction)
```

## **isValidTransaction**

Checks if a Safe transaction can be executed successfully with no errors.

```
const
transactions :
MetaTransactionData [] = [{ // ... }] const
safeTransaction
=
await
protocolKit .createTransaction ({ transactions }) const
isValidTx
=
await
protocolKit .isValidTransaction (safeTransaction)
```

Optionally, some properties can be passed as execution options:

```
const
options :
Web3TransactionOptions
```

```

= { from ,
// Optional gas ,
// Optional gasPrice ,
// Optional maxFeePerGas ,
// Optional maxPriorityFeePerGas // Optional nonce // Optional }
const
options :
EthersTransactionOptions
= { from ,
// Optional gasLimit ,
// Optional gasPrice ,
// Optional maxFeePerGas ,
// Optional maxPriorityFeePerGas // Optional nonce // Optional }
const
isValidTx
=
await
protocolKit .isValidTransaction (safeTransaction , options)

```

## Transaction signatures

### approveTransactionHash

Approves a hash on-chain using the current owner account.

```

const
transactions :
MetaTransactionData [] = [{ // ... }] const
safeTransaction
=
await
protocolKit .createTransaction ({ transactions }) const
txHash
=
await
protocolKit .getTransactionHash (safeTransaction) const
txResponse
=
await
protocolKit .approveTransactionHash (txHash) await
txResponse . transactionResponse ?.wait ()

```

Optionally, some properties can be passed as execution options:

```
const
```

```
options :
```

```
Web3TransactionOptions
```

```
= { from ,
```

```
// Optional gas ,
```

```
// Optional gasPrice ,
```

```
// Optional maxFeePerGas ,
```

```
// Optional maxPriorityFeePerGas // Optional nonce // Optional }
```

```
const
```

```
options :
```

```
EthersTransactionOptions
```

```
= { from ,
```

```
// Optional gasLimit ,
```

```
// Optional gasPrice ,
```

```
// Optional maxFeePerGas ,
```

```
// Optional maxPriorityFeePerGas // Optional nonce // Optional }
```

```
const
```

```
txResponse
```

```
=
```

```
await
```

```
protocolKit .approveTransactionHash (txHash , options)
```

## **signHash**

Signs a hash using the current owner account.

```
const
```

```
transactions :
```

```
MetaTransactionData [] = [{ // ... }] const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createTransaction ({ transactions }) const
```

```
txHash
```

```
=
```

```
await
```

```
protocolKit .getTransactionHash (safeTransaction) const
```

```
signature
```

```
=
```

await

protocolKit .signHash (txHash)

## signTransaction

Returns a newSafeTransaction object that includes the signature of the current owner.

You can use multiple signing methods, such as:

- ETH\_SIGN (eth\_sign
- ): Regular hash signature
- ETH\_SIGN\_TYPED\_DATA\_V4 (eth\_signTypedData\_v4
- ): Typed data signature v4, The default method if no signing method is passed
- ETH\_SIGN\_TYPED\_DATA\_V3eth\_signTypedData\_v3
- : Typed data signature v3
- ETH\_SIGN\_TYPED\_DATAeth\_signTypedData
- : Typed data signature
- SAFE\_SIGNATURE: Signing with another Safe contract as signer

The third parameter (optional) is the preImageSafeAddress. If the preimage is required, this is the address of the Safe that will be used to calculate the preimage. It's a mandatory parameter for 1.3.0 and 1.4.1 contract versions. This is because the safe uses the old EIP-1271 interface, which uses bytes instead of bytes32 for the message; we need to use the pre-image of the message to calculate the message hash. This parameter is used in conjunction with the SAFE\_SIGNATURE signing method.

const

transactions :

MetaTransactionData [] = [{ // ... }] const

safeTransaction

=

await

protocolKit .createTransaction ({ transactions }) const

signedSafeTransaction

=

await

protocolKit .signTransaction (safeTransaction)

Optionally, an additional parameter can be passed to specify a different way of signing:

const

signedSafeTransaction

=

await

protocolKit .signTransaction (safeTransaction ,

SigningMethod . ETH\_SIGN\_TYPED\_DATA\_V4 ) // Default option const

signedSafeTransaction

=

await

protocolKit .signTransaction (safeTransaction ,

SigningMethod . ETH\_SIGN ) const

signedSafeTransaction

=

await

protocolKit.signTransaction (safeTransaction ,  
SigningMethod . SAFE\_SIGNATURE , parentSafeAddress).

## signTypedData

Signs a transaction according to the EIP-712 using the current signer account.

const

transactions :

MetaTransactionData [] = [{ // ... }] const

safeTransaction

=

await

protocolKit.createTransaction ({ transactions }) const

signature

=

await

protocolKit.signTypedData (safeTransaction)

## Owners

### createAddOwnerTx

Returns the Safe transaction to add an owner and optionally change the threshold.

const

params :

AddOwnerTxParams

= { ownerAddress , threshold // Optional. If threshold isn't provided the current threshold won't change. } const

safeTransaction

=

await

protocolKit.createAddOwnerTx (params) const

txResponse

=

await

protocolKit.executeTransaction (safeTransaction) await

txResponse . transactionResponse ?.wait ()

This method can optionally receive the options parameter:

const

options :

SafeTransactionOptionalProps

```
= { ... } const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createAddOwnerTx (params , options)
```

## **createRemoveOwnerTx**

Returns the Safe transaction to remove an owner and optionally change the threshold.

```
const
```

```
params :
```

```
RemoveOwnerTxParams
```

```
= { ownerAddress , newThreshold // Optional. If newThreshold isn't provided, the current threshold will be decreased by one. }
```

```
const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createRemoveOwnerTx (params) const
```

```
txResponse
```

```
=
```

```
await
```

```
protocolKit .executeTransaction (safeTransaction) await
```

```
txResponse . transactionResponse ?.wait ()
```

This method can optionally receive the options parameter:

```
const
```

```
options :
```

```
SafeTransactionOptionalProps
```

```
= { ... } const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createRemoveOwnerTx (params , options)
```

## **createSwapOwnerTx**

Returns the Safe transaction to replace an owner of the Safe with a new one.

```
const
```

```
params :
```

```
SwapOwnerTxParams
```

```
= { oldOwnerAddress , newOwnerAddress } const
```

```
safeTransaction
```

```
=  
await  
protocolKit .createSwapOwnerTx (params) const  
txResponse
```

```
=  
await  
protocolKit .executeTransaction (safeTransaction) await  
txResponse . transactionResponse ?.wait ()
```

This method can optionally receive the options parameter:

```
const  
options :  
SafeTransactionOptionalProps  
= { ... } const  
safeTransaction  
  
=  
await  
protocolKit .createSwapOwnerTx (params , options)
```

## **getOwners**

Returns the list of Safe owner accounts.

```
const  
ownerAddresses  
  
=  
await  
protocolKit .getOwners ()
```

## **getOwnersWhoApprovedTx**

Returns a list of owners who have approved a specific Safe transaction.

```
const  
transactions :  
MetaTransactionData [] = [{ // ... }] const  
safeTransaction  
  
=  
await  
protocolKit .createTransaction ({ transactions }) const  
txHash  
  
=  
await  
protocolKit .getTransactionHash (safeTransaction) const
```



```
ownerAddresses
```

```
=
```

```
await
```

```
protocolKit .getOwnersWhoApprovedTx (txHash)
```

## **isOwner**

Checks if a specific address is an owner of the current Safe.

```
const
```

```
isOwner
```

```
=
```

```
await
```

```
protocolKit .isOwner (address)
```

## **Threshold**

### **createChangeThresholdTx**

Returns the Safe transaction to change the threshold.

```
const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createChangeThresholdTx (newThreshold) const
```

```
txResponse
```

```
=
```

```
await
```

```
protocolKit .executeTransaction (safeTransaction) await
```

```
txResponse . transactionResponse ?.wait ()
```

This method can optionally receive the `options` parameter:

```
const
```

```
options :
```

```
SafeTransactionOptionalProps
```

```
= { ... } const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createChangeThresholdTx (newThreshold , options)
```

### **getThreshold**

Returns the Safe threshold.

```
const
```

threshold

=

await

protocolKit .getThreshold ()

## Safe Guards

### **createDisableGuardTx**

Returns the Safe transaction to disable a Safe Guard.

const

safeTransaction

=

await

protocolKit .createDisableGuardTx () const

txResponse

=

await

protocolKit .executeTransaction (safeTransaction) await

txResponse . transactionResponse ?.wait ()

This method can optionally receive theoptions parameter:

const

options :

SafeTransactionOptionalProps

= { ... } const

safeTransaction

=

await

protocolKit .createDisableGuardTx (options)

### **createEnableGuardTx**

Returns the Safe transaction to enable a Safe Guard.

const

safeTransaction

=

await

protocolKit .createEnableGuardTx (guardAddress) const

txResponse

=

await

protocolKit .executeTransaction (safeTransaction) await

```
txResponse . transactionResponse ?.wait ()
```

This method can optionally receive the options parameter:

```
const
```

```
options :
```

```
SafeTransactionOptionalProps
```

```
= { safeTxGas ,
```

```
// Optional baseGas ,
```

```
// Optional gasPrice ,
```

```
// Optional gasToken ,
```

```
// Optional refundReceiver ,
```

```
// Optional nonce // Optional } const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createEnableGuardTx (guardAddress , options)
```

## **getGuard**

Returns the enabled Safe Guard or 0x address if no guards are enabled.

```
const
```

```
guardAddress
```

```
=
```

```
await
```

```
protocolKit .getGuard ()
```

## **Safe Modules**

### **createDisableModuleTx**

Returns a Safe transaction ready to be signed that will disable a Safe Module.

```
const
```

```
safeTransaction
```

```
=
```

```
await
```

```
protocolKit .createDisableModuleTx (moduleAddress) const
```

```
txResponse
```

```
=
```

```
await
```

```
protocolKit .executeTransaction (safeTransaction) await
```

```
txResponse . transactionResponse ?.wait ()
```

This method can optionally receive the options parameter:

```

const
options :
SafeTransactionOptionalProps
= { ... } const
safeTransaction
=
await
protocolKit .createDisableModuleTx (moduleAddress , options)

```

## **createEnableModuleTx**

Returns a Safe transaction ready to be signed that will enable a Safe Module.

```

const
safeTransaction
=
await
protocolKit .createEnableModuleTx (moduleAddress) const
txResponse
=
await
protocolKit .executeTransaction (safeTransaction) await
txResponse . transactionResponse ?.wait ()

```

This method can optionally receive the options parameter:

```

const
options :
SafeTransactionOptionalProps
= { ... } const
safeTransaction
=
await
protocolKit .createEnableModuleTx (moduleAddress , options)

```

## **getModules**

Returns the list of addresses of all the enabled Safe Modules.

```

const
moduleAddresses
=
await
protocolKit .getModules ()

```

## **isModuleEnabled**

Checks if a specific Safe Module is enabled for the current Safe.

```
const
isEnabled
=
await
protocolKit .isEnabled (moduleAddress)
```

## FallbackHandler

### createDisableFallbackHandlerTx

Returns the Safe transaction to disable the fallback handler.

```
const
safeTransaction
=
await
protocolKit .createDisableFallbackHandlerTx () const
txResponse
=
await
protocolKit .executeTransaction (safeTransaction) await
txResponse . transactionResponse ?.wait ()
```

This method can optionally receive the options parameter:

```
const
options :
SafeTransactionOptionalProps
= { ... } const
safeTransaction
=
await
protocolKit .createDisableFallbackHandlerTx (options)
```

### createEnableFallbackHandlerTx

Returns the Safe transaction to enable the fallback handler.

```
const
safeTransaction
=
await
protocolKit .createEnableFallbackHandlerTx (fallbackHandlerAddress) const
txResponse
=
```

await

protocolKit .executeTransaction (safeTransaction) await

txResponse . transactionResponse ?.wait ()

This method can optionally receive theoptions parameter:

const

options :

SafeTransactionOptionalProps

= { safeTxGas ,

// Optional baseGas ,

// Optional gasPrice ,

// Optional gasToken ,

// Optional refundReceiver ,

// Optional nonce // Optional } const

safeTransaction

=

await

protocolKit .createEnableFallbackHandlerTx (fallbackHandlerAddress , options)

## Messages

### createMessage

Returns a SafeMessage ready to be signed by the owners.

const

rayMessage :

string

|

EIP712TypedData

=

"I am the owner of this Safe" const

message

=

protocolKit .createMessage (rawMessage)

### getSafeMessageHash

Retrieve the Safe message hash of a string, or EIP-712 typed data. It produces the identical hash as invoking the CompatibilityFallbackHandler's getMessageHash method.

const

rawMessage

=

...

// String or EIP-712 typed data const

messageHash

=

hashSafeMessage (rawMessage)

const

safeMessageHash

=

await

protocolKit .getSafeMessageHash (messageHash)

## isValidSignature

Calls the CompatibilityFallbackHandler isValidSignature method (EIP-1271).

It requires two parameters:

- messageHash: The hash of the message
- signature: The signature to be validated or '0x'. You can send as signature one of the following:
  1. An array of SafeSignature. In this case the signatures are concatenated for validation (buildSignatureBytes())
  - 1. The concatenated signatures as string
  - 1. '0x' if you want to validate an onchain message (Approved hash)

The method returns if the signature is valid

const

rawMessage

=

...

// String or EIP-712 typed data const

messageHash

=

hashSafeMessage (rawMessage) const

safeMessageHash

=

await

protocolKit .getSafeMessageHash (messageHash)

const

isValidSignature

=

await

protocolKit .isValidSignature (safeMessageHash , signature) ... const

isValidSignature

=

```

await
protocolKit.isValidSignature (safeMessageHash , [signature1 , signature2]) ... const
isValidSignature
=
await
protocolKit.isValidSignature (safeMessageHash ,
'0x' )

```

## signMessage

Returns a newSafeMessage object that includes the signature of the current owner.

You can use multiple signing methods, such as:

- ETH\_SIGN (eth\_sign
): Regular hash signature
- ETH\_SIGN\_TYPED\_DATA\_V4 (eth\_signTypedData\_v4
): Typed data signature v4, The default method if no signing method is passed
- ETH\_SIGN\_TYPED\_DATA\_V3eth\_signTypedData\_v3
: Typed data signature v3
- ETH\_SIGN\_TYPED\_DATAeth\_signTypedData
: Typed data signature
- SAFE\_SIGNATURE: Signing with another Safe contract as signer

The third parameter (optional) is the preImageSafeAddress. If the preimage is required, this is the address of the Safe that will be used to calculate the preimage. It's a mandatory parameter for 1.3.0 and 1.4.1 contract versions. This is because the safe uses the old EIP-1271 interface, which usesbytes instead ofbytes32 for the message; we need to use the pre-image of the message to calculate the message hash. This parameter is used in conjunction with the SAFE\_SIGNATURE signing method.

```

const
rawMessage :
string
|
EIP712TypedData
=
"I am the owner of this Safe" const
message
=
protocolKit.createMessage (rawMessage) const
signedMessage
=
await
protocolKit.signMessage (message)

```

Optionally, an additional parameter can be passed to specify a different way of signing:

```

const
signedMessage
=
await

```



```
protocolKit .signMessage (signedMessage ,  
SigningMethod . ETH_SIGN_TYPED_DATA_V4 ) // Default option const  
signedMessage  
  
=  
  
await  
  
protocolKit .signMessage (signedMessage ,  
SigningMethod . ETH_SIGN ) const  
signedMessage  
  
=  
  
await  
  
protocolKit .signMessage (signedMessage ,  
SigningMethod . SAFE_SIGNATURE , parentSafeAddress).
```

[Safe Factory Migrating to V1](#)

Was this page helpful?

[Report issue](#)