

# multicall

## Classes

### MultiCaller

Util for executing multi calls against the MultiCallV2 contract

#### Properties

Property	Modifier	Type	Description
address	readonly	string	Address of multicall contract

#### Methods

**getBlockNumberInput()**

getBlockNumberInput ( ) : CallInput < BigNumber

Get the call input for the current block number

Returns

[CallInput](#) <BigNumber

Source

[utils/multicall.ts:133](#)

**getCurrentBlockTimestampInput()**

getCurrentBlockTimestampInput ( ) : CallInput < BigNumber

Get the call input for the current block timestamp

Returns

[CallInput](#) <BigNumber

Source

[utils/multicall.ts:149](#)

**getTokenData()**

getTokenData < T

(erc20Addresses :

string [ ], options ? :

T ) :

Promise < TokenInputOutput < T

[ ]

Multicall for token properties. Will collect all the requested properties for each of the supplied token addresses.

Type parameters

Type parameter T extends undefined |TokenMultiInput

Parameters

Parameter	Type	Description
erc20Addresses	string [ ]	options ? T Defaults to just 'name'

## Returns

Promise <TokenInputOutput <T

[]>

## Source

[utils/multicall.ts:231](#)

**multiCall()**

multiCall < T , TRequireSuccess

( params :

T , requireSuccess ? : TRequireSuccess ) :

Promise < DecoderReturnType < T , TRequireSuccess

Executes a multicall for the given parameters Return values are order the same as the inputs. If a call failed undefined is returned instead of the value.

To get better type inference when the individual calls are of different types create your inputs as a tuple and pass the tuple in. The return type will be a tuple of the decoded return types. eg.

const inputs :

[ CallInput < Awaited < ReturnType < ERC20 [ 'functions' ] [ 'balanceOf' ]

[ 0 ]

, CallInput < Awaited < ReturnType < ERC20 [ 'functions' ] [ 'name' ]

[ 0 ]

]

=

[ { targetAddr : token . address , encoder :

()

=> token . interface . encodeFunctionData ( 'balanceOf' ,

[ " ] ) , decoder :

( returnData :

string )

=> token . interface . decodeFunctionResult ( 'balanceOf' , returnData ) [ 0 ] , } , { targetAddr : token . address , encoder :

()

=> token . interface . encodeFunctionData ( 'name' ) , decoder :

( returnData :

string )

=> token . interface . decodeFunctionResult ( 'name' , returnData ) [ 0 ] , } , ]

const res =

await

multiCaller . call ( inputs )

## Type parameters

Type parameter T extends [CallInput](#) <unknown

[] TRequireSuccess extends boolean

#### Parameters

Parameter Type Description params T requireSuccess ? TRequireSuccess Fail the whole call if any internal call fails

#### Returns

Promise <DecoderReturnType <T ,TRequireSuccess

#### Source

[utils/multicall.ts:197](#)

**fromProvider()**

static

fromProvider ( provider : Provider ) :

Promise < MultiCaller

Finds the correct multicall address for the given provider and instantiates a multicaller

#### Parameters

Parameter Type Description provider Provider

#### Returns

Promise <[MultiCaller](#)

#### Source

[utils/multicall.ts:125](#)

## Type Aliases

### CallInput

type

CallInput < T

: object ; Input to multicall aggregator

#### Type parameters

Type parameter T

#### Type declaration

Member Type Description decoder (returnData :string ) =>T Function to decode the result of the call encoder () =>string Function to produce encoded call data targetAddr string Address of the target contract to be called

#### Source

[utils/multicall.ts:30](#) [Edit this page](#) [Previous Lib](#) [Next Types](#)