RPC API

MetaMask uses the `ethereum.request(args)` method to wrap an RPC API.

The API is based on an interface exposed by all Ethereum clients, along with a growing number of methods that may or may not be supported by other wallets.

> **Tip**
>
> All RPC method requests can return errors. Make sure to handle errors for every call to `ethereum.request(args)`.

> **Try Ethereum Methods**
>
> Visit our API Playground

Table of Contents

Ethereum JSON-RPC Methods

For the Ethereum JSON-RPC API, please see the Ethereum wiki.

Important methods from this API include:

- `eth_accounts`
- `eth_call`
- `eth_getBalance`
- `eth_sendTransaction`
- `eth_sign`

Restricted Methods

MetaMask introduced Web3 Wallet Permissions via EIP-2255. In this permissions system, each RPC method is either *restricted* or *unrestricted*. If a method is restricted, the caller must have the corresponding permission in order to call it. Unrestricted methods, meanwhile, have no corresponding permission. Some of them still rely upon permissions to succeed though (e.g. the signing methods require that you have the `eth_accounts` permission for the signer account), and some require confirmation by the user (e.g. `wallet_addEthereumChain`).

With the exception of MetaMask Flask, the only existing permission is `eth_accounts`, which allows you to access the user's Ethereum address(es). More permissions will be added in the future.

Under the hood, permissions are plain, JSON-compatible objects, with a number of fields that are mostly used internally by MetaMask. The following interface lists the fields that may be of interest to consumers:

```ts
interface Web3WalletPermission {
  // The name of the method corresponding to the permission
  parentCapability: string;

  // The date the permission was granted, in UNIX epoch time
```

If you're interested in learning more about the theory behind this *capability*-inspired permissions system, we encourage you to take a look at EIP-2255 .

`eth_requestAccounts`

> **EIP-1102**
>
> This method is specified by EIP-1102 . It is equivalent to the deprecated `ethereum.enable()` provider API method.
>
> Under the hood, it calls `wallet_requestPermissions` for the `eth_accounts` permission. Since `eth_accounts` is currently the only permission, this method is all you need for now.

Returns

`string[]` - An array of a single, hexadecimal Ethereum address string.

Description

Requests that the user provides an Ethereum address to be identified by. Returns a Promise that resolves to an array of a single Ethereum address string. If the user denies the request, the Promise will reject with a `4001` error.

The request causes a MetaMask popup to appear. You should only request the user's accounts in response to user action, such as a button click. You should always disable the button that caused the request to be dispatched, while the request is still pending.

If you can't retrieve the user's account(s), you should encourage the user to initiate an account request.

Example

```
function connect() {
  ethereum
    .request({ method: 'eth_requestAccounts' })
    .then(handleAccountsChanged)
    .catch((error) => {
      if (error.code === 4001) {
        // EIP-1193 userRejectedRequest error
        console.log('Please connect to MetaMask.');
      } else {
        console.error(error);
      }
    });
}
```

`wallet_getPermissions`

> **Platform Availability**
>
> This RPC method is not yet available in MetaMask Mobile.

Returns

`Web3WalletPermission[]` - An array of the caller's permissions.

Description

Gets the caller's current permissions. Returns a Promise that resolves to an array of `Web3WalletPermission` objects. If the caller has no permissions, the array will be empty.

`wallet_requestPermissions`

> **Platform Availability**

Parameters

- `Array`

  0. `RequestedPermissions` - The requested permissions.

```ts
interface RequestedPermissions {
  [methodName: string]: {}; // an empty object, for future extensibility
}
```

Returns

`Web3WalletPermission[]` - An array of the caller's permissions.

Description

Requests the given permissions from the user. Returns a Promise that resolves to a non-empty array of `Web3WalletPermission` objects, corresponding to the caller's current permissions. If the user denies the request, the Promise will reject with a `4001` error.

The request causes a MetaMask popup to appear. You should only request permissions in response to user action, such as a button click.

Example

```js
document.getElementById('requestPermissionsButton', requestPermissions);

function requestPermissions() {
  ethereum
    .request({
      method: 'wallet_requestPermissions',
      params: [{ eth_accounts: {} }],
    })
    .then((permissions) => {
      const accountsPermission = permissions.find(
```

```
        if (accountsPermission) {
          console.log('eth_accounts permission successfully requested!');
        }
      })
      .catch((error) => {
        if (error.code === 4001) {
          // EIP-1193 userRejectedRequest error
          console.log('Permissions needed to continue.');
        } else {
          console.error(error);
        }
      });
  }
```

Unrestricted Methods

`eth_decrypt`

> **Platform Availability**
>
> This RPC method is not yet available in MetaMask Mobile.

Parameters

- `Array`

0. `string` - An encrypted message.
1. `string` - The address of the Ethereum account that can decrypt the message.

Returns

`string` - The decrypted message.

Description

that resolves to the decrypted message, or rejects if the decryption attempt fails.

See `eth_getEncryptionPublicKey` for more information.

Example

```js
ethereum
  .request({
    method: 'eth_decrypt',
    params: [encryptedMessage, accounts[0]],
  })
  .then((decryptedMessage) =>
    console.log('The decrypted message is:', decryptedMessage)
  )
  .catch((error) => console.log(error.message));
```

`eth_getEncryptionPublicKey`

> **Platform Availability**
>
> This RPC method is not yet available in MetaMask Mobile.

Parameters

- `Array`

  0. `string` - The address of the Ethereum account whose encryption key should be retrieved.

Returns

`string` - The public encryption key of the specified Ethereum account.

Description

The public key is computed from entropy associated with the specified user account, using the `nacl` implementation of the `X25519_XSalsa20_Poly1305` algorithm.

Example

```js
let encryptionPublicKey;

ethereum
  .request({
    method: 'eth_getEncryptionPublicKey',
    params: [accounts[0]], // you must have access to the specified account
  })
  .then((result) => {
    encryptionPublicKey = result;
  })
  .catch((error) => {
    if (error.code === 4001) {
      // EIP-1193 userRejectedRequest error
      console.log("We can't encrypt anything without the key.");
    } else {
      console.error(error);
    }
  });
```

Encrypting

The point of the encryption key is of course to encrypt things. Here's an example of how to encrypt a message using `eth-sig-util` :

```js
const ethUtil = require('ethereumjs-util');
const sigUtil = require('@metamask/eth-sig-util');

const encryptedMessage = ethUtil.bufferToHex(
  Buffer.from(
    JSON.stringify(
      sigUtil.encrypt({
```

```
        version: 'x25519-xsalsa20-poly1305',
      })
    ),
    'utf8'
  )
);
```

`wallet_addEthereumChain`

> **EIP-3085**
>
> This method is specified by EIP-3085 .

Parameters

- `Array`

0. `AddEthereumChainParameter` - Metadata about the chain that will be added to MetaMask.

For the `rpcUrls` and `blockExplorerUrls` arrays, at least one element is required, and only the first element will be used.

```ts
interface AddEthereumChainParameter {
  chainId: string; // A 0x-prefixed hexadecimal string
  chainName: string;
  nativeCurrency: {
    name: string;
    symbol: string; // 2-6 characters long
    decimals: 18;
  };
  rpcUrls: string[];
  blockExplorerUrls?: string[];
  iconUrls?: string[]; // Currently ignored.
}
```

Returns

`null` - The method returns `null` if the request was successful, and an error otherwise.

Description

Creates a confirmation asking the user to add the specified chain to MetaMask. The user may choose to switch to the chain once it has been added.

As with any method that causes a confirmation to appear, `wallet_addEthereumChain` should **only** be called as a result of direct user action, such as the click of a button.

MetaMask stringently validates the parameters for this method, and will reject the request if any parameter is incorrectly formatted. In addition, MetaMask will automatically reject the request under the following circumstances:

- If the RPC endpoint doesn't respond to RPC calls.
- If the RPC endpoint returns a different chain ID when `eth_chainId` is called.
- If the chain ID corresponds to any default MetaMask chains.

MetaMask does not yet support chains with native currencies that do not have 18 decimals, but may do so in the future.

Usage with `wallet_switchEthereumChain`

We recommend using this method with `wallet_addEthereumChain`:

```js
try {
  await ethereum.request({
    method: 'wallet_switchEthereumChain',
    params: [{ chainId: '0xf00' }],
  });
} catch (switchError) {
  // This error code indicates that the chain has not been added to MetaMask.
  if (switchError.code === 4902) {
    try {
      await ethereum.request({
```

```
            {
              chainId: '0xf00',
              chainName: '...',
              rpcUrls: ['https://...'] /* ... */,
            },
          ],
        });
      } catch (addError) {
        // handle "add" error
      }
    }
    // handle other "switch" errors
  }
```

`wallet_switchEthereumChain`

> **EIP-3326**
>
> This method is specified by EIP-3326 .

Parameters

- `Array`

  0. `SwitchEthereumChainParameter` - Metadata about the chain that MetaMask will switch to.

```ts
interface SwitchEthereumChainParameter {
  chainId: string; // A 0x-prefixed hexadecimal string
}
```

Returns

`null` - The method returns `null` if the request was successful, and an error otherwise.

Description

> **Tip**
>
> See above for how to use this method with `wallet_addEthereumChain`.

Creates a confirmation asking the user to switch to the chain with the specified `chainId`.

As with any method that causes a confirmation to appear, `wallet_switchEthereumChain` should **only** be called as a result of direct user action, such as the click of a button.

MetaMask will automatically reject the request under the following circumstances:

- If the chain ID is malformed
- If the chain with the specified chain ID has not been added to MetaMask

`wallet_registerOnboarding`

> **Tip**
>
> As an API consumer, you are unlikely to have to call this method yourself. Please see the Onboarding Library documentation for more information.

Returns

`boolean` - `true` if the request was successful, `false` otherwise.

Description

Registers the requesting site with MetaMask as the initiator of onboarding. Returns a Promise that resolves to `true`, or rejects if there's an error.

This method is intended to be called after MetaMask has been installed, but before the MetaMask onboarding has completed. You can use this method to inform MetaMask that you

Instead of calling this method directly, you should use the `@metamask/onboarding` library.

`wallet_watchAsset`

> **EIP-747**
>
> This method is specified by EIP-747.

Parameters

- `WatchAssetParams` - The metadata of the asset to watch.

```ts
interface WatchAssetParams {
  type: 'ERC20'; // In the future, other standards will be supported
  options: {
    address: string; // The address of the token contract
    'symbol': string; // A ticker symbol or shorthand, up to 5 characters
    decimals: number; // The number of token decimals
    image: string; // A string url of the token logo
  };
}
```

Returns

`boolean` - `true` if the the token was added, `false` otherwise.

Description

Requests that the user tracks the token in MetaMask. Returns a `boolean` indicating if the token was successfully added.

Most Ethereum wallets support some set of tokens, usually from a centrally curated registry of tokens. `wallet_watchAsset` enables web3 application developers to ask their users to track

Example

```js
ethereum
  .request({
    method: 'wallet_watchAsset',
    params: {
      type: 'ERC20',
      options: {
        address: '0xb60e8dd61c5d32be8058bb8eb970870f07233155',
        symbol: 'FOO',
        decimals: 18,
        image: 'https://foo.io/token-image.svg',
      },
    },
  })
  .then((success) => {
    if (success) {
      console.log('FOO successfully added to wallet!');
    } else {
      throw new Error('Something went wrong.');
    }
  })
  .catch(console.error);
```

Mobile Specific RPC Methods

`wallet_scanQRCode`

Parameters

- `Array`

0. `string` - (optional) A regular expression for matching arbitrary QR code strings

Returns

Description

Requests that the user scans a QR code using their device camera. Returns a Promise that resolves to a string, matching either:

1. The regex parameter, if provided
2. An ethereum address, if no regex parameter was provided

If neither condition is met, the Promise will reject with an error.

MetaMask previously introduced this feature per the proposed EIP-945. The functionality was temporarily removed before being reintroduced as this RPC method.

Example

```js
ethereum
  .request({
    method: 'wallet_scanQRCode',
    // The regex string must be valid input to the RegExp constructor, if provid
    params: ['\\D'],
  })
  .then((result) => {
    console.log(result);
  })
  .catch((error) => {
    console.log(error);
  });
```

Edit this page on GitHub

Last Updated: 6/5/2022, 6:08:04 PM

← Provider Migration Guide

Signing Data →