Connect to MetaMask using the Wallet API

You can connect your dapp to users' MetaMask wallets by detecting MetaMask in their browsers and connecting to their accounts. This page provides instructions for connecting to MetaMask using the wallet detection mechanism introduced byEIP-6963 . This approach allows you to detect multiple installed wallets and connect to them without conflicts.

info Learn more about EIP-6963 inWallet interoperability . tip To connect to MetaMask without using EIP-6963, see theCreate a simple dapp tutorial. You can connect to MetaMaskusing third-party libraries ordirectly using Vite .

Connect to MetaMask using third-party libraries

You can connect to MetaMask using the following third-party libraries that support EIP-6963:

- Wagmi 2+
- Web3Modal 3+
- MIPD Store
- RainbowKit
- Web3-Onboard
- ConnectKit

Connect to MetaMask directly using Vite

To connect to MetaMask directly, we recommend implementing support for EIP-6963 using theVite build tool withvanilla TypeScript orReact TypeScript .

Vanilla TypeScript

Follow these steps for creating a vanilla TypeScript project to connect to MetaMask:

1. Create a project

Create a Vite project using the template for vanilla TypeScript:

npm create vite@latest vanilla-ts-6963 -- --template vanilla-ts

2. Set up the project

In your Vite project, updatesrc/vite-env.d.ts with theEIP-6963 interfaces :

vite-env.d.ts ///

interface

EIP6963ProviderInfo

{ rdns :

string uuid :

string name :

string icon :

string }

interface

EIP6963ProviderDetail

{ info : EIP6963ProviderInfo provider : EIP1193Provider }

type

EIP6963AnnounceProviderEvent

=

{ detail :

```
{ info : EIP6963ProviderInfo provider : Readonly < EIP1193Provider
        } }
interface
EIP1193Provider
{ isStatus ? :
boolean host ? :
string path ? :
string sendAsync ? :
( request :
{ method :
string ; params ? :
Array < unknown
} , callback :
( error : Error |
null , response :
unknown )
=>
void )
=>
void send ? :
( request :
{ method :
string ; params ? :
Array < unknown
} , callback :
( error : Error |
null , response :
unknown )
=>
void )
=>
void request :
( request :
{ method :
string params ? :
Array < unknown
        } )
```

=>

Promise < unknown

> } note In addition to the EIP-6963 interfaces, you need aEIP1193Provider interface (defined byEIP-1193), which is the foundational structure for Ethereum wallet providers, and represents the essential properties and methods for interacting with MetaMask and other Ethereum wallets in JavaScript.

3. Updatemain.ts

Updatesrc/main.ts with the following code:

main.ts import

"./style.css" import

{ listProviders }

from

"./providers.ts"

document . querySelector < HTMLDivElement

> ("#app") ! . innerHTML =

` < div

> < div

id

" providerButtons "

> </ div
>
> </ div
>
> `

listProviders (document . querySelector < HTMLDivElement

> ("#providerButtons") !) ThequerySelector finds and returns the first HTML element that matches the CSS selectorapp , and sets itsinnerHTML . You need to include a basic HTML structure with an innerdiv to inject a list of buttons, each representing a detected wallet provider.

You'll create thelistProviders function in the next step, and pass an argument which represents thediv element.

4. Connect to wallets

Create a filesrc/providers.ts with the following code:

providers.ts declare global { interface

WindowEventMap

{ "eip6963:announceProvider" : CustomEvent } }

// Connect to the selected provider using eth_requestAccounts. const connectWithProvider =

async

(wallet : EIP6963AnnounceProviderEvent ["detail"])

=>

{ try

{ await wallet . provider . request ({ method :

"eth_requestAccounts"

}) }

catch

(error)

{ console . error ("Failed to connect to provider:" , error) } }

// Display detected providers as connect buttons. export

function

listProviders (element : HTMLDivElement)

{ window . addEventListener ("eip6963:announceProvider" , (event : EIP6963AnnounceProviderEvent)

=>

{ const button = document . createElement ("button")

button . innerHTML =

` < img

src

" { event . detail . info . icon } "

alt

" { event . detail . info . name } "

/> < div

    { event . detail . info . name } </ div

    `

// Call connectWithProvider when a user selects the button. button . onclick

=

( )

=>

connectWithProvider (event . detail) element . appendChild (button) })

// Notify event listeners and other parts of the dapp that a provider is requested. window . dispatchEvent (new

Event ("eip6963:requestProvider")) } TheconnectWithProvider function connects the user to the selected provider usingeth_requestAccounts . Thewallet object is passed as an argument to the function, indicating the argument type.

ThelistProviders function uses a simplified approach. Instead of mapping and joining an entire block of HTML, it directly passes theevent.detail object to theconnectWithProvider function when a provider is announced.

5. View the project

Run the following command to view and test the Vite project in your browser:

npm run dev

Example

See thevanilla TypeScript example for more information. You can clone the repository and run the example locally usingnpm i && npm run dev .

React TypeScript

Follow these steps for creating a React TypeScript project to connect to MetaMask:

1. Create a project

Create a Vite project using the template for React TypeScript:

```
npm create vite@latest react-ts-6963 -- --template react-ts
```

2. Set up the project

In your Vite project, updatesrc/vite-env.d.ts with theEIP-6963 interfaces :

```
vite-env.d.ts ///

interface

EIP6963ProviderInfo

{ rdns :

string uuid :

string name :

string icon :

string }

interface

EIP6963ProviderDetail

{ info : EIP6963ProviderInfo provider : EIP1193Provider }

type

EIP6963AnnounceProviderEvent

=

{ detail :

{ info : EIP6963ProviderInfo provider : Readonly < EIP1193Provider

      } }

interface

EIP1193Provider

{ isStatus ? :

boolean host ? :

string path ? :

string sendAsync ? :

( request :

{ method :

string ; params ? :

Array < unknown

} , callback :

( error : Error |

null , response :
```

unknown)

=>

void)

=>

void send ? :

(request :

{ method :

string ; params ? :

Array < unknown

} , callback :

(error : Error |

null , response :

unknown)

=>

void)

=>

void request :

(request :

{ method :

string params ? :

Array < unknown

})

=>

Promise < unknown

} note In addition to the EIP-6963 interfaces, you need aEIP1193Provider interface (defined byEIP-1193), which is the foundational structure for Ethereum wallet providers, and represents the essential properties and methods for interacting with MetaMask and other Ethereum wallets in JavaScript.

3. UpdateApp.tsx

Updatesrc/App.tsx with the following code:

App.tsx import

"./App.css" import

{ DiscoverWalletProviders }

from

"./components/DiscoverWalletProviders"

function

App ( )

{ return

(< DiscoverWalletProviders /

) }

export

default App This code renders theDiscoverWalletProviders component that you'll create in the next step, which contains the logic for detecting and connecting to wallet providers.

4. Detect and connect to wallets

In thesrc/components directory, create a componentDiscoverWalletProviders.tsx with the following code:

DiscoverWalletProviders.tsx import

{ useState }

from

"react" import

{ useSyncProviders }

from

"../hooks/useSyncProviders" import

{ formatAddress }

from

"~/utils"

export

const

DiscoverWalletProviders

=

( )

=>

{ const

[selectedWallet , setSelectedWallet]

=

useState < EIP6963ProviderDetail

( ) const

[userAccount , setUserAccount]

=

useState < string

("") const providers =

useSyncProviders ( )

// Connect to the selected provider using eth_requestAccounts. const

handleConnect

=

async

(providerWithInfo : EIP6963ProviderDetail)

```
=>
{ try
{ const accounts =
await providerWithInfo . provider . request ( { method :
"eth_requestAccounts" } )
setSelectedWallet ( providerWithInfo ) setUserAccount ( accounts ?. [ 0 ] ) }
catch
( error )
{ console . error ( error ) } }
// Display detected providers as connect buttons. return
( <
        < h2
        Wallets Detected : < / h2
        < div
        { providers . length
0
? providers ?. map ( ( provider : EIP6963ProviderDetail )
=>
( < button key = { provider . info . uuid } onClick = { ( )
=>
handleConnect ( provider ) }
        < img src = { provider . info . icon } alt = { provider . info . name }
/
        < div
        { provider . info . name } < / div
        < / button
        ) )
: < div
        No Announced Wallet Providers < / div
        } < / div
        < hr /
        < h2
        { userAccount ?
""
:
"No " } Wallet Selected < / h2
        { userAccount && < div
```

< div

< img src = { selectedWallet . info . icon } alt = { selectedWallet . info . name }

/

< div

{ selectedWallet . info . name } < / div

< div

({ formatAddress (userAccount) }) < / div

< / div

< / div

} < /

) } In this code:

- selectedWallet
- is a state variable that holds the user's most recently selected wallet.
- userAccount
- is a state variable that holds the user's connected wallet's address.
- useSyncProviders
- is a custom hook that returns the providers array (wallets installed in the browser).

ThehandleConnect function takes aproviderWithInfo , which is anEIP6963ProviderDetail object. That object is used to request the user's accounts from the provider usingeth_requestAccounts .

If the request succeeds, theselectedWallet anduserAccount local state variables are set.

Then, the component maps over the providers array and renders a button for each detected provider.

Finally, if theuserAccount state variable is not empty, the selected wallet icon, name, and address are displayed.

5. Add React hooks

Create asrc/hooks directory and add astore.ts file with the following code:

hooks/store.ts declare global { interface

WindowEventMap

{ "eip6963:announceProvider" : CustomEvent } }

// An array to store the detected wallet providers. let providers : EIP6963ProviderDetail [ ]

=

[ ]

export

const store =

{ value :

( )

=> providers , subscribe :

(callback :

( )

=>

void)

=>

{ function

onAnnouncement (event : EIP6963AnnounceProviderEvent)

{ if

(providers . map ((p)

=> p . info . uuid) . includes (event . detail . info . uuid)) return providers =

[... providers , event . detail] callback ( ) }

// Listen for eip6963:announceProvider and call onAnnouncement when the event is triggered. window . addEventListener ("eip6963:announceProvider" , onAnnouncement)

// Dispatch the event, which triggers the event listener in the MetaMask wallet. window . dispatchEvent (new

Event ("eip6963:requestProvider"))

// Return a function that removes the event listern. return

( )

=> window . removeEventListener ("eip6963:announceProvider" , onAnnouncement) } , } Also, add a fileuseSyncProviders.ts with the following code to thehooks directory:

hooks/useSyncProviders.ts import

{ useSyncExternalStore }

from

"react" import

{ store }

from

"./store"

export

const

useSyncProviders

=

( )

=> useSyncExternalStore (store . subscribe , store . value , store . value) This hook allows you to subscribe to MetaMask events, read updated values, and update components. It uses thestore.value andstore.subscribe methods defined in thestore.ts hook.

6. Create utility functions

Create asrc/utils directory and add a fileindex.ts with the following code:

index.ts export

const

formatBalance

=

(rawBalance :

string)

=>

{ const balance =

(parseInt (rawBalance)

/

1000000000000000000) . toFixed (2) return balance }

export

const

formatChainAsNum

=

(chainIdHex :

string)

=>

{ const chainIdNum =

parseInt (chainIdHex) return chainIdNum }

export

const

formatAddress

=

(addr :

string)

=>

{ const upperAfterLastTwo = addr . slice (0 ,

2)

+ addr . slice (2) return

` { upperAfterLastTwo . substring (0 ,

5) } ... { upperAfterLastTwo . substring (39) } ` } This is a good place to store utility functions that you might need to reuse throughout your dapp. This example only uses theformatAddress function, but the others might be useful for other applications.

Example

See theReact TypeScript example for more information. You can clone the repository and run the example locally usingnpm i && npm run dev .

Next steps

After connecting to MetaMask directly, you can:

- Detect, add, and switch networks
- .
- Send transactions
- .
- Sign data
- .
- Display tokens, contract methods, and icons in MetaMask
- .

Edit this page