Initializing PnP Web No Modal SDK

After Installation, the next step to use Web3Auth is to Initialize the SDK. However, the Initialization is a two-step process, with an additional two steps for customizations, ie.

- Instantiation of Web3AuthNoModal
- Configuration of Plugins
- (optional)
- Initialization of Web3Auth

Please note that these are the most critical steps where you need to pass on different parameters according to the preference of your project. Additionally, Whitelabeling and Custom Authentication have to be configured within this step, if you wish to customize your Web3Auth Instance.

Instantiating Web3AuthNoModalâ

Import theWeb3AuthNoModal

class from@web3auth/no-modal â

import

{

Web3AuthNoModal

}

from

"@web3auth/no-modal" ;

Assign theWeb3AuthNoModal

class to a variableâ

const web3auth =

new

Web3AuthNoModal (Web3AuthNoModalOptions) ; This Web3AuthNoModal constructor takes an object withWeb3AuthNoModalOptions as input.

Argumentsâ

Web3AuthNoModalOptions

â

- Table
- Interface

Parameter Description chainConfig Custom configuration for your preferred blockchain. Read more about ihere . clientId Your Web3Auth Client ID. You can get it from Web3AuthDashboard under project details. It's a mandatory field of typestring web3AuthNetwork? Defines the Web3Auth network. It acceptsOPENLOGIN_NETWORK_TYPE . useCoreKitKey? Use CoreKit Key to get core kit key. It's an optional field with default value asfalse . sessionTime? It allows developers to configure the session management time. Session Time is in seconds, default is 86400 seconds which is 1 day.sessionTime can be max 7 days. uiConfig? WhiteLabel options for web3auth. It helps you define custom UI, branding, and translations for your brand app. It takesWhiteLabelData as a value. storageKey? Setting to "local" will persist social login session across browser tabs. privateKeyProvider? Private key provider for your chain namespace. It takesIBaseProvider as a value. enableLogging Setting to true will enable logs. Default value isfalse . interface

Web3AuthNoModalOptions

{ /* * *Client id for web3auth. * You can obtain your client id from the web3auth developer dashboard. * You can set any random string for this on localhost.* / clientId :

string ; /* * *custom chain configuration for chainNamespace * * @defaultValue mainnet config of provided chainNamespace*/

chainConfig :

Partial < CustomChainConfig

&

Pick < CustomChainConfig ,

"chainNamespace"

; /* * setting to true will enable logs * * @defaultValue false*/ enableLogging ? :

boolean ; /* * setting to "local" will persist social login session accross browser tabs. * * @defaultValue "local"* storageKey ? :

"session"

|

"local" ; /* * sessionTime (in seconds) for idToken issued by Web3Auth for server side verification. * @defaultValue 86400 * * Note: max value can be 7 days (86400 * 7) and min can be 1 day (86400) */ sessionTime ? :

number ; /* * Web3Auth Network to use for the session & the issued idToken * @defaultValue mainnet* web3AuthNetwork ? :

OPENLOGIN_NETWORK_TYPE ; /* * Uses core-kit key with web3auth provider * @defaultValue false*/ useCoreKitKey ? :

boolean ; /* * WhiteLabel options for web3auth*/ uiConfig ? :

WhiteLabelData ; /* * Private key provider for your chain namespace*/ privateKeyProvider ? :

IBaseProvider < string

; }

Adding a Custom Chain Configurationâ

chainConfig

â

- Table
- Type Declarations

Parameter Description chainNamespace The namespace of your preferred chain. CheckoutProviders SDK Reference for understanding RPC Calls. It acceptsChainNamespaceType as a value. chainId The chain id of the selected blockchain in hexstring format. rpcTarget * RPC Target URL for the selectedchainNamespace * &chainId * . * We provide a default RPC Target for certain blockchains, but due to congestion it might be slow hence it is recommended to provide your own RPC Target URL. wsTarget Web socket target URL for the chain instring . displayName Display Name for the chain instring . blockExplorerUrl Blockchain's explorer URL instring . (eg:https://etherscan.io) ticker Default currency ticker instring for the network (e.g:ETH) tickerName Name for currency ticker instring (e.g:Ethereum) decimals? Number of decimals innumber for the currency ticker (e.g:18) logo Logo for the chain. isTestnet? Defines whether the network is testnet or not. declare

const

CHAIN_NAMESPACES :

{ readonly

EIP155 :

"eip155" ; readonly

SOLANA :

"solana" ; readonly

OTHER :

"other" ; } ;

declare

type

ChainNamespaceType

=

(typeof

CHAIN_NAMESPACES) [keyof

typeof

CHAIN_NAMESPACES] ; declare

type

CustomChainConfig

=

{ chainNamespace :

ChainNamespaceType ; /* * *The chain id of the chain*/ chainId :

string ; /* * *RPC target Url for the chain*/ rpcTarget :

string ; /* * *web socket target Url for the chain*/ wsTarget ? :

string ; /* * *Display Name for the chain*/ displayName :

string ; /* * *Url of the block explorer*/ blockExplorerUrl :

string ; /* * *Default currency ticker of the network (e.g: ETH)*/ ticker :

string ; /* * *Name for currency ticker (e.g:Ethereum)* / tickerName :

string ; /* * *Number of decimals for the currency ticker (e.g: 18)*/ decimals ? :

number ; /* * *Logo for the chain*/ logo :

string ; /* * *Whether the network is testnet or not*/ isTestnet ? :

boolean ; } ;

Returnsâ

- Object
- : The web3auth instance with all its methods and events.

Instantiating a Providerâ

For getting the appropriate key for your selected chain, you need to instantiate a provider. This provider is used to make calls to the selected blockchain. Currently, Web3Auth exposes the following provider packages to be integrated within the SDKs:

- EthereumPrivateKeyProvider
- : For Ethereum and EVM compatible chains (secp256k1 private key)
- SolanaPrivateKeyProvider
- : For Solana Blockchain (ed25519 private key)
- XRPLPrivateKeyProvider
- : For XRP Ledger
- CommonPrivateKeyProvider
- : For other blockchains (secp256k1 private key)

Using these providers, you can export the private keys for the respective curves of the selected blockchain. You can use these keys to even create manual providers for other blockchains supporting the curves/ have conversion packages for their blockchain curves.

- ETH

- Solana
- XRPL
- Other Chains

Usage import

{

EthereumPrivateKeyProvider

}

from

"@web3auth/ethereum-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

EthereumPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0x1" , rpcTarget :

"https://rpc.ankr.com/eth" , displayName :

"Ethereum Mainnet" , blockExplorer :

"https://etherscan.io" , ticker :

"ETH" , tickerName :

"Ethereum" , } , } , }) ; Usage import

{

SolanaPrivateKeyProvider

}

from

"@web3auth/solana-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

SolanaPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x1" , rpcTarget :

"https://rpc.ankr.com/solana" , displayName :

"Solana Mainnet" , blockExplorer :

"https://explorer.solana.com/" , ticker :

"SOL" , tickerName :

"Solana" , } , } , }) ; Usage import

{

XrplPrivateKeyProvider

}

from

"@web3auth/xrpl-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

XrplPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . OTHER , chainId :

"0x1" , rpcTarget :

"https://s.altnet.rippletest.net:51234" , displayName :

"XRPL" , blockExplorer :

"https://testnet.xrpl.org" , ticker :

"XRP" , tickerName :

"XRP" , } , } , }) ; Usage import

{

CommonPrivateKeyProvider

}

from

"@web3auth/base-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

CommonPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . OTHER , chainId :

"0x1" , rpcTarget :

https://rpc.target.url , displayName :

"Display Name" , blockExplorer :

"https://chain.explorer.link" , ticker :

"TKR" , tickerName :

"Ticker Name" , } , } , }) ;

Whitelabelingâ

uiConfig

â

For customizing the redirect screens while logging in and constructing the key, you need to pass onWhiteLabelData in theuiConfig parameter to theuiConfig property of Web3Auth.

tip This is just one of the aspects of whitelabeling you can achieve with Web3Auth. To know more in-depth about how you can Whitelabel your application with Web3Auth, have a look at ourWhitelabeling SDK Reference .

Exampleâ

const uiConfig :

WhiteLabelData

=

{ appName :

"My App" , appURL :

"https://example.com" , logoLight :

"https://example.com/logo-light.png" , logoDark :

"https://example.com/logo-dark.png" , defaultLanguage :

LANGUAGES . en , mode :

"light" , useLogoLoader :

true , theme :

{ primary :

"#FF0000" , secondary :

"#00FF00" , warning :

"#FFA500" , } , tncLink :

"https://example.com/terms" , privacyPolicy :

"https://example.com/privacy" , } ;

const web3auth =

new

Web3AuthNoModal ({ clientId :

"" ,

// Get your Client ID from the Web3Auth Dashboard chainConfig , web3AuthNetwork :

WEB3AUTH_NETWORK . SAPPHIRE_MAINNET , privateKeyProvider , uiConfig , }) ;

Exampleâ

const chainConfig :

CustomChainConfig

=

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0x1" , displayName :

"Ethereum Mainnet" , blockExplorer :

"https://etherscan.io" , ticker :

"ETH" , tickerName :

"Ethereum" , rpcTarget :

"https://rpc.ankr.com/eth" ,

// This is the mainnet RPC we have added, please pass on your own endpoint while creating an app } ; const privateKeyProvider =

new

EthereumPrivateKeyProvider ({ config :

{ chainConfig }

}) ; const web3auth =

new

Web3AuthNoModal ({ clientId :

"" ,

// Get your Client ID from the Web3Auth Dashboard chainConfig , web3AuthNetwork :

WEB3AUTH_NETWORK . SAPPHIRE_MAINNET , privateKeyProvider , uiConfig :

{ theme :

{ primaryColor :

"#FF0000" , secondaryColor :

"#00FF00" , textColor :

"#0000FF" , } , } , }) ;

Configuring Adaptersâ

An adapter is a pluggable package that implements anIAdapter interface for a wallet within Web3Auth. An adapter can be plugged in and out of the web3auth modal. Each adapter exposes the provider on successful user login that can be used to invoke RPC calls on the wallet and connected blockchain. You can configure and use any of the adapters provided by Web3Auth, by just configuring them while initializing Web3AuthNoModal.

Instantiating a Providerâ

For getting the appropriate key for your selected chain, you need to instantiate a provider. This provider is used to make calls to the selected blockchain. Currently, Web3Auth exposes the following provider packages to be integrated within the SDKs:

- EthereumPrivateKeyProvider
- : For Ethereum and EVM compatible chains (secp256k1 private key)
- SolanaPrivateKeyProvider
- : For Solana Blockchain (ed25519 private key)
- XRPLPrivateKeyProvider
- : For XRP Ledger
- CommonPrivateKeyProvider
- : For other blockchains (secp256k1 private key)

Using these providers, you can export the private keys for the respective curves of the selected blockchain. You can use these keys to even create manual providers for other blockchains supporting the curves/ have conversion packages for their blockchain curves.

- ETH
- Solana
- XRPL
- Other Chains

Usage import

{

EthereumPrivateKeyProvider

}

from

"@web3auth/ethereum-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

EthereumPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0x1" , rpcTarget :

"https://rpc.ankr.com/eth" , displayName :

"Ethereum Mainnet" , blockExplorer :

"https://etherscan.io" , ticker :

"ETH" , tickerName :

"Ethereum" , } , } , }) ; Usage import

{

SolanaPrivateKeyProvider

}

from

"@web3auth/solana-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

SolanaPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . SOLANA , chainId :

"0x1" , rpcTarget :

"https://rpc.ankr.com/solana" , displayName :

"Solana Mainnet" , blockExplorer :

"https://explorer.solana.com/" , ticker :

"SOL" , tickerName :

"Solana" , } , } , }) ; Usage import

{

XrplPrivateKeyProvider

}

from

"@web3auth/xrpl-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

XrplPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . OTHER , chainId :

"0x1" , rpcTarget :

"https://s.altnet.rippletest.net:51234" , displayName :

"XRPL" , blockExplorer :

"https://testnet.xrpl.org" , ticker :

"XRP" , tickerName :

"XRP" , } , } , }) ; Usage import

{

CommonPrivateKeyProvider

}

from

"@web3auth/base-provider" ; import

{

CHAIN_NAMESPACES

}

from

"@web3auth/base" ;

const privateKeyProvider =

new

CommonPrivateKeyProvider ({ config :

{ / *pass the chain config that you want to connect with. all chainConfig fields are required*/ chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . OTHER , chainId :

"0x1" , rpcTarget :

https://rpc.target.url , displayName :

"Display Name" , blockExplorer :

"https://chain.explorer.link" , ticker :

"TKR" , tickerName :

"Ticker Name" , } , } , }) ;

Configuring Openlogin Adapterâ

The default adapter of Web3Auth is theopenlogin-adapter . This adapter is a wrapper around theopenlogin library from Web3Auth and enables the social login features. For customising features of the main Web3Auth flow, likeWhitelabel ,Custom Authentication , etc. you need to customise the Openlogin Adapter.

tip Checkout theopenlogin-adapter SDK Reference for more details on different configurations you can pass for

customisations.

Whitelabelingâ

whiteLabel

â

For customising the redirect screens while logging in and constructing the key, you need to pass onwhiteLabel configurations to theadapterSettings property of theopenlogin-adapter .

tip This is just one of the aspects of whitelabeling you can achieve with Web3Auth. To know more in depth about how you can Whitelabel your application with Web3Auth, have a look at ourWhitelabeling SDK Reference .

Exampleâ

import

{

OpenloginAdapter

}

from

"@web3auth/openlogin-adapter" ;

const openloginAdapter =

new

OpenloginAdapter ({ adapterSettings :

{ clientId ,

//Optional - Provide only if you haven't provided it in the Web3Auth Instantiation Code network :

"sapphire_mainnet" ,

// Optional - Provide only if you haven't provided it in the Web3Auth Instantiation Code uxMode :

"popup" , whiteLabel :

{ appName :

"W3A Heroes" , appUrl :

"https://web3auth.io" , logoLight :

"https://web3auth.io/images/web3auth-logo.svg" , logoDark :

"https://web3auth.io/images/web3auth-logo---Dark.svg" , defaultLanguage :

"en" ,

// en, de, ja, ko, zh, es, fr, pt, nl, tr mode :

"dark" ,

// whether to enable dark mode. defaultValue: auto theme :

{ primary :

"#00D1B2" , } , useLogoLoader :

true , } , } , privateKeyProvider , }) ; web3auth . configureAdapter (openloginAdapter) ;

Custom Authenticationâ

loginConfig

â

With Web3Auth, you have the option to configure logins using your own authentication services. For adding your own authentication, you have to first configure your verifiers in the Web3Auth Dashboard. Have a look at ourCustom Authentication Documentation for configuring that first.

Custom Authentication in Web3Auth is supported by the Openlogin Adapter, which is the default adapter for the Web3Auth SDK. For this, you need to configure theloginConfig parameter in theadapterSettings of theopenlogin-adapter package.

tip Refer to theCustom Authentication Documentation for more information.

Exampleâ

Since we're using the@web3auth/no-modal , ie. the Plug and Play No Modal SDK, theloginConfig can include custom JWT-based authentication as well. This way, we can use any of our preferred login providers and further setup their configs while logging the user in and passing over theextraLoginOptions in theconnectTo function. Read more about this in theCustom Authentication Guide .

```
import

{

OpenloginAdapter

}

from

"@web3auth/openlogin-adapter" ;

const openloginAdapter =

new

OpenloginAdapter ( { adapterSettings :

{ clientId ,

//Optional - Provide only if you haven't provided it in the Web3Auth Instantiation Code network :

"sapphire_mainnet" ,

// Optional - Provide only if you haven't provided it in the Web3Auth Instantiation Code uxMode :

"popup" , loginConfig :

{ jwt :

{ verifier :

"YOUR-VERIFIER-NAME-ON-WEB3AUTH-DASHBOARD" , typeOfLogin :

"jwt" , clientId :

"YOUR-CLIENTID-FROM-LOGIN-PROVIDER" , } , } , } , privateKeyProvider , } ) ;

web3auth . configureAdapter ( openloginAdapter ) ;
```

Configuring External Wallet Adaptersâ

configureAdapter(ADAPTER)

â

To configure an adapter, create the instance of adapter by using its corresponding package and pass the returnedadapter instance inconfigureAdapter function.

tip Refer to theAdapters documentation to know more deeply about what adapters are available and how to configure them.

Exampleâ

If you want to configure the Torus EVM Wallet Adapter

- Import theTorusWalletAdapter
- from@web3auth/torus-evm-adapter
- package
- Create an instance of the adapter along with the configuration
- Pass the returned instance in toweb3auth.configureAdapter

```
import

{

TorusWalletAdapter

}

from

"@web3auth/torus-evm-adapter" ;

const torusAdapter =

new

TorusWalletAdapter ( { adapterSettings :

{ clientId ,

//Optional - Provide only if you haven't provided it in the Web3Auth Instantiation Code buttonPosition :

"bottom-left" , } , loginSettings :

{ verifier :

"google" , } , initParams :

{ buildEnv :

"testing" , } , chainConfig :

{ chainNamespace :

CHAIN_NAMESPACES . EIP155 , chainId :

"0x1" , rpcTarget :

"https://rpc.ankr.com/eth" ,

// This is the mainnet RPC we have added, please pass on your own endpoint while creating an app displayName :

"Ethereum Mainnet" , blockExplorerUrl :

"https://etherscan.io/" , ticker :

"ETH" , tickerName :

"Ethereum" , logo :

"https://images.toruswallet.io/eth.svg" , } , } ) ;

web3auth . configureAdapter ( torusAdapter ) ;
```

Subscribing the Lifecycle Eventsâ

Subscribing to events help you trigger responses based on the status of the connection of the user. An adapter emits certain events likeCONNECTED ,CONNECTING andDISCONNECTED etc during login lifecycle of a user. For example, you can use this to show an error message, if the user is not connected to the network. Generally, this is not a required step and should be done only if needed in particular cases.

info This step is totally optional. If you don't want to use any plugins, feel free to skip this section. tip If you're using theuxMode: "redirect" option within youropenlogin-adapter configuration, you need to subscribe to the event to handle the logging in implicitly. This is because, when redirected to a different application, the app state is not updated as per the login status. Using a lifecycle method to check this, one can easily handle the login status within the constructor function.

on(EVENT, CALLBACK)

â

Web3Auth provides the following lifecycle event to check the login status:

Adapter Eventsâ

- Table
- Type Declarations

Event Trigger with@web3auth/base package Trigger without package Description ADAPTER_DATA_UPDATED ADAPTER_EVENTS.ADAPTER_DATA_UPDATED "adapter_data_updated" Adapter data is updated within the dApp NOT_READY ADAPTER_EVENTS.NOT_READY "not_ready" Adapter is not yet ready for login READY ADAPTER_EVENTS.READY "ready" Adapter is ready for login CONNECTING ADAPTER_EVENTS.CONNECTING "connecting" User is connecting to the dApp/ login process is in progress CONNECTED ADAPTER_EVENTS.CONNECTED "connected" User is logged in and connected with the dApp DISCONNECTED ADAPTER_EVENTS.DISCONNECTED "disconnected" User is logged out and disconnected from the dApp ERRORED ADAPTER_EVENTS.ERRORED "errored" There has been some error in connecting the user to the dApp declare

const

ADAPTER_EVENTS :

{ readonly

ADAPTER_DATA_UPDATED :

"adapter_data_updated" ; readonly

NOT_READY :

"not_ready" ; readonly

READY :

"ready" ; readonly

CONNECTING :

"connecting" ; readonly

CONNECTED :

"connected" ; readonly

DISCONNECTED :

"disconnected" ; readonly

ERRORED :

"errored" ; } ;

Exampleâ

import

{

ADAPTER_EVENTS

}

from

"@web3auth/base" ;

// subscribe to lifecycle events emitted by web3auth const

subscribeAuthEvents

=

(web3auth :

Web3Auth)

=>

{ web3auth . on (ADAPTER_EVENTS . CONNECTED ,

(data :

CONNECTED_EVENT_DATA)

=>

{ console . log ("connected to wallet" , data) ; // web3auth.provider will be available here after user is connected }) ;
web3auth . on (ADAPTER_EVENTS . CONNECTING ,

( )

=>

{ console . log ("connecting") ; }) ; web3auth . on (ADAPTER_EVENTS . DISCONNECTED ,

( )

=>

{ console . log ("disconnected") ; }) ; web3auth . on (ADAPTER_EVENTS . ERRORED ,

(error)

=>

{ console . log ("error" , error) ; }) ; web3auth . on (ADAPTER_EVENTS . ERRORED ,

(error)

=>

{ console . log ("error" , error) ; }) ; } ;

Configuring Pluginsâ

Plugins are essentially extensions to the core functionality of Web3Auth, allowing you to add additional features to your dApp. These features can be used to extend the UI functionalities, making your integration more interoperable, and a lot more, even having the functionality to be customised extremely and to your liking.

info This step is totally optional. If you don't want to use any plugins, feel free to skip this section. Currently Web3Auth supports the following two plugins:

- @web3auth/torus-wallet-connector-plugin
- @web3auth/solana-wallet-connector-plugin

tip Learn about adding plugins to your Web3Auth instancehere .

showWalletConnectScanner()

â

Shows the Wallet Connect Scanner to connect with dApps having Wallet Connect login option. This is useful for interoperability with dApps having Wallet Connect login option.

Exampleâ

import

{

WalletServicesPlugin

}

from

"@web3auth/wallet-services-plugin" ;

const walletServicesPlugin =

new

WalletServicesPlugin ( ) ; web3auth . addPlugin (walletServicesPlugin) ;

// Add the plugin to web3auth

await walletServicesPlugin . showWalletConnectScanner ( ) ;

initiateTopup()

â

Shows the TopUp modal to select local currency and amount to top up the wallet.

Exampleâ

import

{

WalletServicesPlugin

}

from

"@web3auth/wallet-services-plugin" ;

const walletServicesPlugin =

new

WalletServicesPlugin ( ) ; web3auth . addPlugin (walletServicesPlugin) ;

// Add the plugin to web3auth

await walletServicesPlugin . showCheckout ( ) ;

// Opens the TopUp modal

Initializing Web3Authâ

init()

â

The final step in the whole initialization process is the initialize theWeb3AuthNoModal instance, ie.web3auth in our case.

This is done by calling theinit function of theweb3auth instance we created above.

await web3auth . init ( ) ; This is a simple function, that doesn't take any input, nor does return anything. It just makes sure that theweb3auth instance is initialized and ready for the user to log in. Edit this page Previous Install Next Usage