Snaps quickstart

Get started creating your own Snap. Use the@metamask/create-snap starter kit to initialize a Snap monorepo project built with TypeScript and React. See the following video demo:

Prerequisites

- MetaMask Flask installed
- A text editor (for example,VS Code
-)
- Node
- version 20.11 or later
- Yarn

Create the project

Create a new Snap project using the@metamask/create-snap starter kit by running:

yarn create @metamask/snap your-snap-name or

npx @metamask/create-snap your-snap-name or

npm create @metamask/snap your-snap-name SeeSnaps files to learn about your Snap project files.

Start the Snap

From the root of the newly created project, install the project dependencies using Yarn:

yarn

install You may get a warning like the following:

@lavamoat/allow-scripts has detected dependencies without configuration. explicit configuration required. run "allow-scripts auto" to automatically populate the configuration. You can fix this by running the following command:

yarn run allow-scripts auto Start the development server:

yarn start You are now serving the Snap athttp://localhost:8080 and its front-end dapp athttp://localhost:8000 .

Connect to the Snap

On the front-end dapp, select theConnect button and the MetaMask Flask extension pops up and requires you to approve the Snap's permissions.

Once connected, select theSend message button to display a custom message within a confirmation dialog in MetaMask Flask.

Customize the Snap

Open the project in a text editor. You can customize your Snap by editingindex.ts in thepackages/snap/src folder.

index.ts contains an example request that uses thesnap_dialog method to display a custom confirmation screen:

index.ts import

type

{ OnRpcRequestHandler }

from

"@metamask/snaps-sdk" ; import

{ panel , text }

from

"@metamask/snaps-sdk" ;

/ * Handle incoming JSON-RPC requests, sent through wallet_invokeSnap. * * @param args - The request handler arguments as an object. * @param args.origin - The origin of the request, e.g., the website that invoked the Snap. * @param args.request - A validated JSON-RPC request object. * @returns The result of snap_dialog. * @throws If the request method is not valid for this Snap. /* export

const onRpcRequest :

OnRpcRequestHandler

=

async

({ origin , request , })

=>

{ switch

(request . method)

{ case

"hello" : return snap . request ({ method :

"snap_dialog" , params :

{ type :

"confirmation" , content :

panel ([text (Hello, ** { origin } **!) , text ("This custom confirmation is just for display purposes.") , text ("But you can edit the Snap source code to make it do something, if you want to!" ,) ,]) , } , }) ; default : throw

new

Error ("Method not found.") ; } } ; Edit the text in anytext() component and select theReconnect button on the front-end to re-install the Snap.

note MetaMask Flask automatically re-installs locally hosted Snaps whenever it receives a connection request for them. The next time you select theSend message button, you see the updated text in the confirmation dialog.

You've now successfully connected, installed, interacted with, and customized your Snap!

Next steps

- To learn more about the Snaps system, reviewfundamental Snaps concepts
- and try theSnaps tutorials
- .
- To implement specific features and use cases, see theSnaps feature guides
- .
- To follow best practices for developing a Snap, see theSnaps how-to guides
- .

Edit this page