Localization

You can localize your Snap to display your Snap's user interface (UI) text and textual metadata (such as title and description) in the user's language.

Steps

1. Get the user's language

In your Snap's code, determine the user's language by using thesnap_getLocale API method. To callsnap_getLocale , first request the required permission by adding it to theinitialPermissions field in your manifest file:

snap.manifest.json { "initialPermissions" :

{ "snap_getLocale" :

{ } } } Your Snap can then callsnap_getLocale to get the user's language code (for example,en ores).

2. Localize the Snap's UI

Use the user's language code to localize your Snap's UI.

First, create locale files namedsnap/locales/.json , containing content in the following format:

en.json { "locale" :

"en" , "messages" :

{ "hello" :

{ "message" :

"Hello, world!" , "description" :

"The message that is returned when the hello method is called." } } } You can then use these files in a localization module. The following is an example module:

import da from

"../locales/da.json" ; import en from

"../locales/en.json" ; import nl from

"../locales/nl.json" ;

// Default language, to be used if there is not a valid translation in // the requested locale. const

FALLBACK_LANGUAGE : Locale =

"en" ;

export

const locales =

{ da : da . messages , en : en . messages , nl : nl . messages , } ;

export

type

Locale

=

keyof

typeof locales ;

export

async

function

getMessage (id :

keyof

(typeof locales) [Locale])

{ const locale =

(await snap . request ({ method :

"snap_getLocale"

}))

as Locale ; const

{ message }

= locales [locale] ?. [id]

?? locales [FALLBACK_LANGUAGE] [id] ;

return message ; } You can then use thegetMessage function in your Snap's code to localize a message, which uses English as the default if the user's locale isn't available.

The following is an example of usinggetMessage in a Snap's RPC request handler:

index.ts import

{ rpcErrors }

from

"@metamask/rpc-errors" ; import

type

{ OnRpcRequestHandler }

from

"@metamask/snaps-sdk" ;

import

{ getMessage }

from

"./locales" ;

export

const onRpcRequest :

OnRpcRequestHandler

=

async

({ request })

=>

{ switch

(request . method)

{ case

"hello" : return

await

getMessage ("hello") ;

default : throw rpcErrors . methodNotFound ({ data :

{ method : request . method } , }) ; } } ;

3. Localize the Snap's manifest file

The Snapmanifest file contains textual metadata such asproposedName anddescription that you can localize to display in the user's language.

The following is an example of a localized manifest file:

snap.manifest.json { "version" :

"1.1.1" , "description" :

"{{ description }}" , "proposedName" :

"{{ name }}" , "source" :

{ "shasum" :

"XXX" , "locales" :

["locales/da.json" , "locales/en.json" , "locales/nl.json"] } , "initialPermissions" :

{ "snap_getLocale" :

{ } } , "manifestVersion" :

"0.1" } To localize your manifest file, add all available locales to thesource.locales section, listing the locale files you created whenlocalizing the Snap's UI . The example locale file provided inStep 2 follows MetaMask's format for manifest localization.

For each field that needs to be localized, replace its text with{{}} .

When displaying this metadata to the user, MetaMask searches for the proper locale file and replaces the template string with its translation. If it doesn't find the user's locale, MetaMask uses English (en) as the default.

Example

See the@metamask/localization-example-snap package for a full example of localizing a Snap.

Edit this page