@cowprotocol/app-data

AppData schema definitions

These schemas are used in the data encoded onappData field for CowProtocol orders.

For more details, checkthe docs .

Installation

yarn add @cowprotocol/app-data

Usage

import

{ MetadataApi }

from

'@cowprotocol/app-data'

export

const metadataApi =

new

MetadataApi ( )

const appCode =

'YOUR_APP_CODE' const environment =

'prod' const referrer =

{ address :

REFERRER_ADDRESS

}

const quote =

{ slippageBips :

'0.5'

}

// Slippage percent, it's 0 to 100 const orderClass =

{ orderClass :

'market'

}

// "market" | "limit" | "liquidity"

const appDataDoc =

await metadataApi . generateAppDataDoc ({ appCode , environment , metadata :

{ referrer , quote , orderClass , } , })

const

{ cid , appDataHex , appDataContent }

=

await metadataApi . appDataToCid (appDataDoc)

// 🐮 You should use appDataHex as the appData value in the CoW Order. "cid" Identifies the metadata associated to the
IPFS

// You can derive the CID from the appDataHex of any order const actualCid =

await metadataApi . appDataHexToCid (appDataHex) console . log (cid === actualCid)

// Should be true

// You can derive the appDataHex from the CID of any order const actualAppDatahex =

await metadataApi . appDataHexToCid (cid) console . log (appDataHex === actualAppDatahex)

// Should be true

// You can retrieve the JSON document from the CID // NOTE: for this to work, someone needs to upload the document to API does it, but anyone could upload it too) const actualAppDoc =

await

fetchDocFromCid (cid) expect (actualAppDoc) . toBeEqual (appDataDoc)

// You can also retrieve the JSON from the appDataHex const actualAppDoc2 =

await

fetchDocFromAppDataHex (appDataHex) expect (actualAppDoc2) . toBeEqual (appDataDoc)

Schemas

Schemas are exposed as json files, where the version is the file name:

// Getting the version v0.4.0 const schema =

require ('@cowprotocol/app-data/schemas/v0.4.0.json')

// Now you can for example run validation against a schema

Type definitions

There are also type definitions

import

{ v0_4_0 }

from

'@cowprotocol/app-data'

// Note: this example is function

createAppDataV0_4_0 (appCode : v0_4_0 . AppCode ,

metadata : v0_4_0 . Metadata) : v0_4_0 . AppDataRootSchema

{ return

{ version :

'0.4.0' , appCode , metadata , } }

Constants

The latest version names are exposed as constants

import

{ LATEST_APP_DATA_VERSION , LATEST_QUOTE_METADATA_VERSION , LATEST_REFERRER_METADATA_VERSION , }

from

'@cowprotocol/app-data'

Utils

Get appData schema

To get a schema definition by version

import

{ getAppDataSchema }

from

'@cowprotocol/app-data'

const schema =

getAppDataSchema ('0.1.0') It'll throw if the version does not exist

Validate appDataDoc

To validate a document, pass it tovalidateAppDataDoc . It'll return an object with a boolean indicatingsuccess anderrors , if any. The version to validate against will be taken from the doc itself.

import

{ validateAppDataDoc }

from

'@cowprotocol/app-data'

let doc =

{

version :

'0.4.0' ,

metadata :

{ }

}

let result =

await

validateAppDataDoc (doc) console . log (result)

// { success: true }

doc

{

version :

'0.0.0' ,

metadata :

{ }

}

result

await

validateAppDataDoc (doc) // Contrary togetAppDataSchema, invalid or non-existing schemas won't throw console . log (result)

// { success: false, errors: 'AppData version 0.0.0 doesn\'t exist'}

Contribute

Fork the repo so you can create a new PR. Then:

1. Add a new version for the schema using the semver
2. convention
3. Just duplicate the latest version i.e.src/schemas/.json
4. tosrc/schemas/.json
5. If you are adding a new meta-data
6. We create one directory per schema, so we can keep track of all versions. Create the directory and initial schema definition:/v0.1.0.json
7. Add it to the main schema you just created in step 1:"ref": "./v0.1.0.json#"
8. .

9. Example:https://github.com/cowprotocol/app-data/pull/44/files#diff-7f7a61b478245dfda004f64bd68ac55ef68cbeb5d6d90d77e1cdbd2b7e1212b8R56
10. If you are modifying an existing meta-data
11. Version it using thesemver
12. convention
13. You will need to create the new file for the meta-data schema/.json
14. Update it in the main schema you just created in step 1: Set it to": { "ref": "/.json#" }
15. Modify thecompile.ts
16. script
17. Add the exported constant with the latest version in, and the new metadata:* For examplehttps://github.com/cowprotocol/app-data/pull/44/commits/aeef8a58e7bbd2a53664ce396011cb157a18406d
18. Generate the typescript types
19. Runyarn build
20. Make a test focusing on the new or modified meta-data:
21. https://github.com/cowprotocol/app-data/pull/44/files#diff-e755a2ecce42f09829d5c7dc1de8853d1d00ef56eaadc2709601c87b9be8ddfbR556
22. Don't forget to use the right version of the schema in your test:https://github.com/cowprotocol/app-data/pull/44/files#diff-e755a2ecce42f09829d5c7dc1de8853d1d00ef56eaadc2709601c87b9be8ddfbR11
23. Create the PR and document it together with the motivation for the changes

Previous TwapStruct Next Exports