Creating Your First Intent

This guide will walk you through creating your first intent: a recurring swap intent.Suggest Edits

This guide usesviem for wallet signatures andaxios for HTTP requests. You may replace these with other equivalent packages of your choice, such asethers.js orfetch respectively.

We've also opted to usedotenv to hide our secrets, as we will be using both an EOA private key and a Brink API key.

Private Key Security

Note: We at Brink DO NOT encourage storing private keys in .env files. This is for demonstration purposes only. Please use a more secure method of storing your private keys in production. The steps outlined in this tutorial can be done either within a new codebase or within an existing project.

To view and play with the demo script that this guide is based on and more, please visit ourpublic guides repository on GitHub .

Setup

We'll start by installing our packages and setting up the JavaScript code file. In the file, we will create one main() function that runs our script end-to-end. We'll also load our secrets from our .env file.

terminal npm install viem axios dotenv script.js const axios = require ('axios')const viem = require ('viem')require ('dotenv').config ()const main = async ()=> {// ... }main ()

Brink API Key

This guide makes use of the Brink API which requires an API key. Pleasecontact us on Discord to get yours!

1. Constructing the Intent

The intent we'll be creating in this guide is arecurring swap intent , AKA dollar-cost averaging (DCA). We want to swap 5,000 USDC for ETH (WETH) approximately once a week for 3 months or until we cancel the intent early. We'll also incentivize the solver with 2.5% of the swap.

The first step in creating this intent is using Brink's domain-specific language (DSL). The DSL structure for one intent consists of an object with multiple fields that define the guardrails of the intent, which include the actions , conditions and replay fields.

Brink Intents DSL

Inside of our main() function, we will add our Brink DSL object, declaring our recurring swap intent with it's parameters and conditions.

script.js const axios = require ('axios')const viem = require ('viem')require ('dotenv').config ()const main = async ()=> {const myRecurringIntent = {actions : [{type :'marketSwap' ,owner :'0xc0ffee' ,// EOA public address of signer tokenInAmount :500000000 ,// 500 USDC (6 decimals) tokenIn :'0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48' ,// USDC tokenOut :'0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2' ,// WETH fee :2.5 // incentivize solver with 2.5% of the swap }],conditions : [{id :'1234567890' ,// unique ID for condition, perferably randomly generated type :'interval' ,interval :50_000 ,// ~50,000 blocks are built every 7 days maxIntervals :12 // 7 days * 12 === 12 weeks or 3 months }],replay : {nonce :'123' ,// TODO, must make API request for this value, shown below runs :'UNTIL_CANCELLED' } } }main () The myRecurringIntent intent object is made up of 3 fields: actions , conditions , and replay .

The actions field is an array of objects that define the outcome(s) of our intent, in our case, a 500 USDC -> WETH market swap. We also set a 2.5% fee that the solver may take as an incentive to fulfill our intent.

Similarly, the conditions field is an array of objects that define the conditions that the actions may be executed against. Here, we expect the actions to run at a block interval of 50,000 blocks for 12 intervals to meet our "once a week for 3 months" intent condition.

Finally, the replay field defines the replayability for our intent. Since replayability is necessary for any recurring intent or transaction, we allow it by setting the runs field to UNTIL_CANCELLED , theoretically allowing our intent to be replayed

forever (thankfully, our maxIntervals value would prevent our intent fromactually being run forever).

We must also specify the replay.nonce value, which requires a quick request to the Brink API.

Fetching the Nonce Value

To fetch the correct nonce value, we must make a GET request to the /signers/nonces/v1 endpoint (replacing with the public address of the EOA we'll be using to sign and submit this intent). Once we have the response data, we can plug it in to our intent object.

We'll start by putting our Brink API key into our .env file for security, then loading it into our code as a request header in axios with an x-api-key field.

.env BRINK_API_KEY =script.js const axios = require ('axios')const viem = require ('viem')require ('dotenv').config ()const main = async ()=> {const nonceRes = await axios .get ('https://api.brink.trade/signers/0xc0ffee/nonces/v1' , {headers : {'x-api-key' :process .env .BRINK_API_KEY , } })const myRecurringIntent = {actions : [{type :'marketSwap' ,tokenInAmount :500000000 ,tokenIn :'USDC' ,tokenOut :'ETH' ,fee :2.5 }],conditions : [{type :'interval' ,interval :50_000 ,maxIntervals :12 }],replay : {nonce :parseInt (nonceRes .data .nonces [0]),// nonce value from API response runs :'UNTIL_CANCELLED' } } }main ()

2. Compiling the Intents DSL

Now that we've defined our intent using the Brink Intents DSL, we must compile it into a Brink Declaration and construct theEIP-712 signature payload before signing it. Thankfully, there is a single Brink API endpoint that quickly compiles the DSLand provides the signature payload for us: /intents/compile/v1 .

What is a Declaration ?

A Declaration is a new concept introduced in the Brink protocol that allows for multiple Intents to be submitted in bulk with only one signature. Each time you sign an off-chain message (e.g. EIP-712 signature) on Brink, a Declaration is created, and may contain as many Intents as the user or developer would like.

A Segment (actions, conditions, replay values) is the lowest level of granularity in the Brink protocol. Segments make up Intents , and Intents make up Declarations . This empowers developers to create truly composable and complex intents quickly and easily. In ourcompile request, we must define a few fields to properly compile the intent. Since this is a GET request, we will pass our data as query parameters.

Params for the /intents/compile/v1 request include:

- chainId
- : The chain ID of the network you are using (we are using Ethereum Mainnet,
- 1
-).
- signer
- : The address of the account that will be signing the intent.
- signatureType
- : We'll be using the value
- EIP712
- for this guide.
- include
- : An array of extra data to include in the API response.
- declaration
- : The intent object we created earlier (also accepts an array of multiple intents).

Using the params object in axios, we can define our parameters for this call as such:

script.js const axios = require ('axios')const viem = require ('viem')require ('dotenv').config ()const main = async ()=> {// ... nonce request const myRecurringIntent = {/ *intent definition* / }const compileRes = await axios .get ('https://api.brink.trade/intents/compile/v1' , {headers : {'x-api-key' :process .env .BRINK_API_KEY , },params : {chainId :1 ,signer :'0xc0ffee' ,signatureType :'EIP712' ,include : ['required_transactions'],declaration :JSON .stringify (myRecurringIntent) } }) }main () We must pass required_transactions as an item in the include array. By doing so, the Brink API will respond with any transactions that are required to be finalizedbefore our intent can be fulfilled by solvers. In our case, a token approval transaction is required.

Once submitted, the /intents/compile/v1 response data will be structured as such:

compile response data // response from /intents/compile/v1 {declaration : {... },eip712Data : {types : {... },domain : {... },value : {...

},hash :'0x' },eip1271Data : {message :'Not implemented' },hash :'0x ,requiredTransactions : [{owner :'' ,spender :'' ,currentAllowance :'0' ,requiredAllowance :'500000000' ,token : {... },minTx : {... },maxTx : {... } }] } This data will be put to use for the next 2 steps: approving your Brink Proxy to spend tokens, and signing the EIP-712 signature.

3. Approving Your Brink Proxy

Before signing and submitting our declaration, we mustapprove the signer's Brink Proxy contract to transfer the tokens involved in the intent. For more info on how the Proxy contracts work in the Brink protocol, please reach out onDiscord .

We can get all info about the approval transaction in the requiredTransactions field of the API response above.

compile response data {//... other response fields requiredTransactions : [{owner :'' ,spender :'' ,currentAllowance :'0' ,requiredAllowance :'500000000' ,token : {address :'0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48' ,// USDC token address // ... other info },minTx : {... },maxTx : {... } }] } To simplify this guide, an easy way to quickly approve tokens is by using the Etherscan "Write Contract" (sometimes "Write Proxy Contract") section of the token contract page.

From the owner account, you should approve the spender address to spend the tokens (the token contract address found at token.address). Wehighly recommend approving for an unlimited amount of tokens (max uint256 value), but the requiredAllowance value in the response is the minimum amount of tokens you must approve.

Approving Tokens Programmatically

If you'd like to run the approval transaction programmatically or within a web app, you may construct the transaction using either the minTx or the maxTx objects in the requiredTransactions response array. The minTx and maxTx contain the exact executable calldata for the approval transaction for both the minimum allowance amount and the maximum allowance amount respectively.

4. Signing the Declaration

Now that we've approved our Brink Proxy to spend our tokens, we can finally sign the intent (declaration). To do so, we must construct the EIP-712 typed data that we received from the /intents/compile/v1 endpoint. These values can be found in the eip712Data field of the response. We also must set the primaryType field to MetaDelegateCall .

We can prepare viem to sign by adding our private key to our .env file, then loading it into our code as a walletClient . Once our walletClient is created, we can construct and sign the EIP-712 signature using the signTypedData() method, passing in our values from the eip712Data field and our walletClient.account object.

.env

.env file

BRINK_API_KEY

SIGNER_PRIVATE_KEY

Private Key Security

Note: We at Brink DO NOT encourage storing private keys in .env files. This is for demonstration purposes only. Please use a more secure method of storing your private keys in production. script.js const axios = require ('axios')const viem = require ('viem')require ('dotenv').config ()const main = async ()=> {// ... nonce request const myRecurringIntent = {/ *intent declaration* / }const compileRes = await axios .get ('https://api.brink.trade/intents/compile/v1' , {// ... request config })const {eip712Data }= compileRes .data const walletClient = viem .createWalletClient ({account :privateKeyToAccount (0x{ process .env .PRIVATE_KEY }), })const declarationSignature = await walletClient .signTypedData ({account :walletClient .account ,types :eip712Data .types ,domain :eip712Data .domain ,message :eip712Data .value ,primaryType :'MetaDelegateCall' }) }main ()

5. Submitting the Intent

Now that we have our declaration signature using viem , we can finally submit it to the Brink API. To do so, we must make a POST request to the /intents/submit/v1 endpoint. The data we pass to this endpoint is similar to the data we passed to the /intents/compile/v1 endpoint, but it's passed as the POST request body, rather than query parameters.

The POST request body for the /intents/compile/v1 request includes:

- chainId
- : The chain ID of the network you are using (we are using Ethereum Mainnet,
- 1
-).
- signer
- : The address of the account that will be signing the intent.
- signatureType
- :
- EIP712
- same as before
- declaration
- : Thecompiled
- declaration given to us from the compile request step.
- signature
- : The EIP-712 signature hash we previously signed.

script.js const axios = require ('axios')const viem = require ('viem')require ('dotenv').config ()const main = async ()=> {// ... nonce request and intent declaration const compileRes = await axios .get ('https://api.brink.trade/intents/compile/v1' , {// ... request config })// ... walletClient from previous step const declarationSignature = await walletClient .signTypedData ({// ... signTypedData config })const submitRes = await axios .post ('https://api.brink.trade/intents/submit/v1' , {chainId :1 ,signer :signerPublicAddress ,signatureType :'EIP712' ,declaration :compileRes .data .declaration ,// compiled declaration from compile request signature :declarationSignature // signature hash from signTypedData }, {headers : {'x-api-key' :process .env .BRINK_API_KEY } }) }main () Once the signed Brink Declaration is submitted to the Brink API, it will be added to the Brink Intentpool and the intent we created will be available for solvers to fulfill once the conditions are met.

Conclusion

Congratulations! You've successfully created your first Brink Intent and submitted it as a Declaration. Your final code file should look something like the code block below.

For a full, runable script, please visitscript.js file in our public guides GitHub repo.

script.js const axios = require ('axios')const viem = require ('viem')require ('dotenv').config ()const main = async ()=> {const nonceRes = await axios .get ('https://api.brink.trade/signers/0xc0ffee/nonces/v1' , {headers : {'x-api-key' :process .env .BRINK_API_KEY , } })const myRecurringIntent = {actions : [{type :'marketSwap' ,owner :'0xc0ffee' ,// EOA public address of signer tokenInAmount :500000000 ,tokenIn :'0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48' ,// USDC tokenOut :'0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2' ,// WETH fee :2.5 // incentivize solver with 2.5% of the swap }],conditions : [{id :'1234567890' ,// unique ID for condition, perferably randomly generated type :'interval' ,interval :50_000 ,// ~50,000 blocks are built every 7 days maxIntervals :12 // 7 days * 12 === 12 weeks or 3 months }],replay : {nonce :parseInt (nonceRes .data .nonces [0]),// nonce value from API response runs :'UNTIL_CANCELLED' } }const compileRes = await axios .get ('https://api.brink.trade/intents/compile/v1' , {headers : {'x-api-key' :process .env .BRINK_API_KEY , },params : {chainId :1 ,signer :'0xc0ffee' ,signatureType :'EIP712' ,include : ['required_transactions'],declaration :JSON .stringify (myRecurringIntent) } })const {eip712Data }= compileRes .data const walletClient = viem .createWalletClient ({account :privateKeyToAccount (0x{ process .env .PRIVATE_KEY }), })const declarationSignature = await walletClient .signTypedData ({account :walletClient .account ,types :eip712Data .types ,domain :eip712Data .domain ,message :eip712Data .value ,primaryType :'MetaDelegateCall' })const submitRes = await axios .post ('https://api.brink.trade/intents/submit/v1' , {chainId :1 ,signer :signerPublicAddress ,signatureType :'EIP712' ,declaration :compileRes .data .declaration ,// compiled declaration from compile request signature :declarationSignature // signature hash from signTypedData }, {headers : {'x-api-key' :process .env .BRINK_API_KEY } })console .log (submitRes .data) }main () Updated6 months ago

Protocol Overview Building a Brink Solver Did this page help you?Yes No *Table of Contents * *Setup * *1. Constructing the Intent * * *Brink Intents DSL * * *Fetching the Nonce Value * *2. Compiling the Intents DSL * *3. Approving Your Brink Proxy * * *Approving Tokens Programmatically * *4. Signing the Declaration * *5. Submitting the Intent * *Conclusion