Building a Brink Solver

A high level guide on building your own solverSuggest Edits

Introduction

Brink depends on solvers to ensure that intents are settled at competitive pricing and in a timely manner. To encourage this, solvers are incentivized with the possibility of extracting tokens from the overall transaction, after the intent signer's requirements are satisfied. A solver will monitor the Brink API for new intents, and work to find the best solution in the shortest possible time to solve the intent, winning the extractable value from the intent.

This guide will teach you how to build your own solver that will compete to settle Brink intents. We will take you through the technical steps required to get you up and running!

Fetching Intents from the API

Before you begin solving intents, you will first need to fetch them using the Brink API. To do this you will need a Brink API Key, you can obtain this from the team by asking for one in ourdiscord or reaching out toBrinkTrade on twitter

In order to get a list of intents, you will need to query theFind Intent Declarations endpoint. In the example below, we are querying this endpoint and sorting by most recently created and filtered by Polygon intents only. To do this, we set sortDir=desc and chainId=137 .

JSX https ://api.brink.trade/intents/declarations/find/v1?chainId=137&sortDir=desc You may be wondering why the endpoint is called declarations . Adeclaration in Brink is a signed message that can contain one or more executable intents. All intents within the declaration will have the same signer.

Here is an example declaration that will be returned from this endpoint, which we will use for future reference:

{ "chainId": 137, "createdAt": "2024-02-12T22:23:17.742Z", "declaration": { "afterCalls": [], "beforeCalls": [], "intents": [{ "segments": [{ "functionName": "requireBitNotUsed", "params": { "index": "0", "value": "8589934592" }, "requiresUnsignedCall": false, "data": "0x000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000000400000000000000000000000000000000 }, { "functionName": "swap01", "params": { "signer": "0xcEd90a58Ba727ABa04bb8C11a3b40a1c866b36F0", "tokenIn": { "id": "0", "address": "0x7ceB23fD6bC0adD59E62ac25578270cFf1b9f619", "standard": 0, "idsMerkleRoot": "0x0000000000000000000000000000000000000000000000000000000000000000", "disallowFlagged": false }, "tokenOut": { "id": "0", "address": "0x0b3F868E0BE5597D5DB7fEB59E1CADBb0fdDa50a", "standard": 0, "idsMerkleRoot": "0x0000000000000000000000000000000000000000000000000000000000000000", "disallowFlagged": false }, "inputAmount": { "params": ["10000000000000000"], "paramTypes": [{ "name": "amount", "type": "uint256" }], "contractName": "FixedSwapAmount01", "contractAddress": "0x19baae0c65ef0bc41e0fa160babe605f2c1d7052", "paramsBytesData": "0x000000000000000000000000000000000000000000000000002386f26fc10000" }, "outputAmount": { "params": ["10000000000000000", "8015073528051394333", "53448107", "1600", "800", "3000", "-3800", "0x74bc6232d7b5cf8db6dd5cb8264e9cca6beed605", "0x000000000000000000000000312692e9cadd3ddaace2e112a4e36397bd2f18a000000000000000000000000000000000000000000000000000000000000000078"], "paramTypes": [{ "name": "oppositeTokenAmount", "type": "uint256" }, { "name": "blockIntervalId", "type": "uint64" }, { "name": "firstAuctionStartBlock", "type": "uint128" }, { "name": "auctionDelayBlocks", "type": "uint128" }, { "name": "auctionDurationBlocks", "type": "uint128" }, { "name": "startPercentE6", "type": "int24" }, { "name": "endPercentE6", "type": "int24" }, { "name": "priceX96Oracle", "type": "address" }, { "name": "priceX96OracleParams", "type": "bytes" }], "contractName": "BlockIntervalDutchAuctionAmount01", "contractAddress": "0x935be545c327602aa17f9184deab53be59440769", "paramsBytesData": "0x000000000000000000000000000000000000000000000000002386f26fc100000000000000000000000000000000000000000000000000006f3b42e87823e71d00000000000000000000000000000000 }, "solverValidator": "0xea8cb64b49a29db66e84a795dc46ebfa9b10b120" }, "requiresUnsignedCall": true, "data": "0x000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000000400000000000000000000000000000000 }, { "functionName": "blockInterval", "params": { "id": "8015073528051394333", "initialStart": "0", "maxIntervals": "1", "intervalMinSize": "1600" }, "requiresUnsignedCall": false, "data": "0x000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000000400000000000000000000000000000000 }] }], "segmentsContract": "0xb83235717a536a57063035a97fcb40a0b0ac0759", "data": "0x4f899fb2000000000000000000000000000000000000000000000000000000000000004000000000000000000000000000000000000000000000000000000000000000840000000000000000000000 }, "declarationContract": "0xd313e04a4bd954e33c6daabd7f1b277b8087a948", "eip712Data": { "hash": "0xf58f905b2ca9f040745ce7f0c75deed0c2be30b1965e5d0d51a76f99e2b8f704", "types": { "metaDelegateCall": [{ "name": "to", "type": "address" }, { "name": "data", "type": "bytes", "calldata": true }] }, "value": { "to": "0xd313e04a4bd954e33c6daabd7f1b277b8087a948", "data": "0x4f899fb2000000000000000000000000000000000000000000000000000000000000004000000000000000000000000000000000000000000000000000000000000000840000000000000000000000 }, "domain": { "name": "BrinkAccount", "chainId": 137, "version": "1", "verifyingContract": "0x0C7b3A72Bd1e7E6eDdb768b447CCeE8aA0A5c5dA" } }, "hash": "0xf58f905b2ca9f040745ce7f0c75deed0c2be30b1965e5d0d51a76f99e2b8f704", "nonces": [{ "bit": { "index": "0", "value": "8589934592" }, "nonce": "34", "segmentIndex": 0, "intentIndex": 0 }], "signature": "0x2fcb3b06ffecc25ed9718fb3b70b7d272e880a2c5488ddc71f3a2225db464e050a947518ea540b94580b5f0c2e69c3dbc85e6351beab6e0687a4465443456e8d1c", "signatureType": "EIP712", "signer": "0xced90a58ba727aba04bb8c11a3b40a1c866b36f0", "source": "recurring_swap_widget", "state": { "intents": [{ "intentIndex": 0, "statuses": ["MAX_INTERVALS_EXCEEDED"] }] }, "tokens": [{ "token": { "address": "0x7ceB23fD6bC0adD59E62ac25578270cFf1b9f619", "standard": 0, "idsMerkleRoot": "0x0000000000000000000000000000000000000000000000000000000000000000", "id": "0", "disallowFlagged": false }, "segmentIndex": 1, "tokenParam": "tokenIn", "isInput": true, "amount": "10000000000000000", "intentIndex": 0 }, { "token": { "address": "0x0b3F868E0BE5597D5DB7fEB59E1CADBb0fdDa50a", "standard": 0, "idsMerkleRoot": "0x0000000000000000000000000000000000000000000000000000000000000000", "id": "0", "disallowFlagged": false }, "segmentIndex": 1, "tokenParam": "tokenOut", "isInput": false, "intentIndex": 0 }], "transactions": [{ "hash": "0xa2b386e43dfb9a156e37f871bfc8bb4f996a9bb76d6b2f959a95e10ea8348143", "intentIndex": 0, "status": "failed", "time": "2024-02-12T22:24:38.202" }, { "hash": "0xd3d9ca9af13461dd9e4512206518d01b907935d58640bc5815c0767cc2157970", "intentIndex": 0, "status": "failed", "time": "2024-02-12T22:26:10.301" }, { "hash": "0xa3b08aafcc578d52a7b74c37fe8bfc0d8e23f4040f3c5c1e715e99b8e4e67857", "intentIndex": 0, "status": "succeeded", "time": "2024-02-12T22:27:22.347" }] }

Segments

Once you have the declaration data you are ready to begin solving. The above declaration only contains one intent, which contains 3 segments. ASegment is the fundamental building block of a Brink intent. Each segment has a correspondingsmart contract function . In order to solve an intent, each segment's smart contract function has to run without reverting. Most segments fall into a category we call conditionals . These run simple EVM state checks like ensuring the intent hasn't been cancelled or hasn't expired. Outside of conditionals, we have a segments called actions . Action segments require you as a solver to generate solution calldata in order to successfully run the segment.

The action segment we will use in this example is calledswap01 . This segment supports fixed and dynamic input and output for token swaps. The example intent uses a descending output dutch auction swap.

How to Evaluate Conditional Segments

Segments within an intent should be evaluated off-chain by solvers in order to determine if the intent is settleable or not. Here are some conditional segments and details on how to evaluate them

BlockInterval

This segment is used to limit settlement of an intent to a specified number of blocks. For example, 1 intent settlement per 100 blocks.

The BlockInterval conditional can be evaluated using theBlock Interval API endpoint. This endpoints response data will contain four fields

JSX const {intervalReady ,maxIntervalsExceeded ,currentBlockNumber ,state }= res .data If intervalReady is true and maxIntervalsExceeded is false , the interval is ready, you can continue the intent evaluation.

if maxIntervalsExceeded is true , the number of max intervals has been exceeded, you can stop all further evaluation on this intent.

if intervalReady is false and maxIntervalsExceeded is false , the interval is not ready, but will be ready at some point. You can determine when this time will be by using the chain's average block time, state.start - the start of the interval, intervalMinSize - the size of the interval in blocks (obtainable from the segment data), and currentBlockNumber

RequireBitUsed

This segment requires that a signed bit has been set on-chain.

The RequireBitUsed conditional can be evaluated using theUse Bit API endpoint. This endpoint's response data will contain a boolean field called success . If success is true , the bit is not used, so you should retry the intent later, if success is false , the bit is used, and you can continue evaluating the intent.

RequireBitNotUsed

This segment requires that a signed bit has NOT been set on-chain.

The RequireBitNotUsed conditional can be evaluated using theUse Bit API endpoint. This endpoint's response data will contain a boolean field called success . If success is false , the bit is used, meaning this intent is likely cancelled or completed and should not be retried. If success is true , the bit is not used, and you can continue evaluating the intent.

RequireBlockMined

This segment requires that the current block number is greater than or equal to a signed block number.

The RequireBlockMined conditional can be evaluated using theRequire Block Mined API endpoint. This endpoint's response data will contain a boolean field called success . If success is true , the block is not mined, so you should retry the intent later, if success is false , the block is mined, and you can continue evaluating the intent.

RequireBlockNotMined

This segment requires that the current block number is less than a signed block number.

The RequireBlockNotMined conditional can be evaluated using the Require Block Not Mined API endpoint. This endpoint's response data will contain a boolean field called success . If success is false , the block is mined, meaning this intent can no longer be solved and should not be retried. If success is true , the block is not mined, and you can continue evaluating the intent.

RequireUint256UpperBound

This segment requires that a uint256 value on-chain is less than a signed "upper bound" value.

The RequireUint256UpperBound conditional can be evaluated using the Required Uint256 Upper Bound API endpoint. This endpoint's response data will contain a boolean field called success . If success is false , then the uint value is below the upper bound, and the solver should retry the intent later. If success is true , the uint value has hit the upper bound, and the solver should continue evaluating the intent.

RequireUint256LowerBound

This segment requires that a uint256 value on-chain is greater than a signed "lower bound" value.

The RequireUint256LowerBound conditional can be evaluated using the Require Uint256 Lower Bound API endpoint. This endpoint's response data will contain a boolean field called success . If success is false , then the uint value is above the lower bound, and the solver should retry the intent later. If success is true , the uint value has hit the lower bound, and the solver should continue evaluating the intent.

Solving the Swap01 Segment

After all conditionals in the intent are evaluated and your solver determines that these conditions are solvable on-chain, you will need to solve the user's signed action. In this example, the action is the swap01 segment.

The swap01 segment determines the rules around the inputs and outputs required by the signer. We have simplified this to a single API endpoint that will return how many tokens the signer will require for the transaction to run without revert. Before calling this endpoint, you will want to ensure that the signer has the funds and has sufficient approval to their Brink proxy account for the transaction to succeed.

1. Check the signer's balance
2.
   1. This is a straightforward check on the
3.
   1. tokenIn
4.
   1. balance of the
5.
   1. signer
6.
   1. In the above declaration, the intent's signer is
7.
   1. 0xced90a58ba727aba04bb8c11a3b40a1c866b36f0
8.
   1. and the tokenIn is
9.
   1. 0x7ceB23fD6bC0adD59E62ac25578270cFf1b9f619
10. Check the proxy account approval
11.
    1. You will need to ensure the proxy account has the approval necessary to transfer for the swap.
12.
    1. You can obtain the proxy account address by using the Brink SDK
13.
    1. .
14.
    1. JSX
15.
    1. import
16.
    1. {getSignerAccount
17.
    1. }from
18.
    1. '@brinkninja/sdk'
19.
    1. const
20.
    1. accountAddress
21.
    1. =
22.
    1. getSignerAccount
23.
    1. ({signer
24.
    1. :'0xced90a58ba727aba04bb8c11a3b40a1c866b36f0'
25.
    1. })

Next you will want to call the swap01 API endpoint. Below is a reference of how to populate the required inputs for swap01. The parameters come directly from the segment data, except the chain ID, which comes from the declaration data.

JSX const chainId = declaration .chainId const {signer ,tokenIn ,tokenOut ,inputAmount ,outputAmount ,solverValidator }= swap01Params //Swap 01 Segment data const swapQueryParams = new URLSearchParams ({chainId ,inputAmountContract :inputAmount .contractAddress ,inputAmountParams :inputAmount .paramsBytesData ,outputAmountContract :outputAmount .contractAddress ,outputAmountParams :outputAmount .paramsBytesData ,owner :signer ,solverValidator ,tokenIn :tokenIn .address ,tokenOut :tokenOut .address })const swapApiUrl = new URL ('/segments/swap01/v1' ,'https://api.brink.trade')swapRes = await get (swapApiUrl .toString ()+ '?' + swapQueryParams .toString (), {headers : {"x-api-key" :BRINK - API - KEY } })const minTokenOutAmount = swapRes .data .output .amount This important piece of information you will want from this api call is the minTokenOutAmount this is the minimum token amount in denomination of tokenOut that the signer needs to receive from the transaction in order for swap01 not to revert.

Submitting your Solution

Once you have the minTokenOutAmount , you will need to generate a solution transaction where the signer of the intent will receive that amount of tokenOut tokens. You will need to decide how to do this yourself. This will be the secret sauce of your solver and where you can gain your competitive edge. You will want to create an adapter contract that will help you to interact with the protocol. Here is Brink's Adapter for reference.

Once you have your solution transaction ready for submission, you have a few more steps.

1. Assemble the unsigned swap data hash using your adapter and calldata
2. JSX
3. import
4. {unsignedSwapDataHash
5. ,IdsProof
6. }from
7. '@brinkninja/sdk'
8. const
9. unsignedSwapHash
10. =
11. await

12. unsignedSwapDataHash
13. ({recipient
14. :YOUR_ADAPTER_ADDRESS
15. ,tokenInIdsProof
16. :new
17. IdsProof
18. (),tokenOutIdsProof
19. :new
20. IdsProof
21. (),callData
22. : {targetContract
23. :YOUR_ADAPTER_ADDRESS
24. ,data
25. :YOUR_CALLDATA
26. }
27. })
28. Sign the swap hash with your solver's private key. Also ensure your solver address is added as a valid solver by the intent signer by checking the SolverValidator
29. contract on the relevant network.
30. import { ethers, Wallet } from 'ethers'
31. const solverSigner = new Wallet(YOUR_SOLVER_PRIVATE_KEY);
32. const signedSwapHash = await solverSigner.signMessage(ethers.getBytes(unsignedSwapHash))
33. Assemble the unsigned swap call using the signedSwapHash from the previous step
34. import { unsignedSwapData, IdsProof } from '@brinkninja/sdk'
35. const unsignedSwapCall = await unsignedSwapData({
36. recipient: YOUR_ADAPTER_ADDRESS,
37. tokenInIdsProof: new IdsProof(),
38. tokenOutIdsProof: new IdsProof(),
39. callData: {
40. targetContract: YOUR_ADAPTER_ADDRESS,
41. data: YOUR_CALLDATA
42. },
43. signature: signedSwapHash
44. })
45. Check if the proxy account has been deployed for the signer. The following code assumes you have an ethers provider setup.
46. JSX
47. import
48. {accountCode
49. }from
50. '@brinkninja/sdk'
51. const
52. {method
53. ,params
54. }=
55. accountCode
56. ({signer
57. :declaration
58. .signer
59. })const
60. accountCodeResp
61. =
62. await
63. provider
64. .send
65. (method
66. ,params
67.)let
68. deployAccount
69. =
70. true
71. if
72. (accountCodeResp
73. !==
74. '0x'
75.) {deployAccount
76. =
77. false
78. }
79. Generate the final transaction data using the variables from the previous steps
80. JSX
81. import
82. {executeIntent
83. ,SignedDeclaration
84. }from
85. '@brinkninja/sdk'
86. // Turn the declaration data into a signed declaration
87. const
88. {signer
89. ,chainId
90. ,signatureType
91. ,signature
92. ,declaration
93. ,declarationContract
94. }=
95. declaration
96. const
97. signedDeclaration
98. =
99. new
100. SignedDeclaration
101. ({signer
102. ,chainId
103. ,signatureType
104. ,signature
105. ,declaration
106. ,declarationContract
107. })const
108. executeStrategyTx
109. =
110. await
111. executeIntent
112. ({signedDeclaration
113. ,intentIndex
114. ,unsignedCalls
115. : [unsignedSwapCall
116.],deployAccount
117. })
118. Send the transaction!
119. JSX
120. tx
121. =

```
await
signer
.sendTransaction
({to
:executeStrategyTx
.to
,data
:executeStrategyTx
.data
})
```

Conclusion

This guide has gone over the fundamentals of creating a Brink solver.

Please reach out to the Brink team ondiscord ortwitter with any questions or comments. Updated21 days ago

Creating Your First Intent Getting Started Did this page help you?Yes No