How to create your own inflator contract and submit compressed user operations to Pimlico's bundler

This guide assumes you are already familiar with the process of creating and submitting normal user operations to bundlers. If you are not, we recommend you start withtutorial 1 of our library, permissionless.js .

This guide walks you through how to create your own inflator contract, compress your user operations and submit them to Pimlico's bundler to achieve significant gas cost savings for your users.

In order to submit compressed user operations to Pimlico's bundler, you need to:

- Write and deploy your inflator contract implementing theIOpInflator
- interface
- Get the Inflator contract address whitelisted by Pimlico
- Start submitting your compressed user operations using thepimlico_sendCompressedUserOperation
- JSON-RPC method

Write and deploy your inflator contract

First, we will write a simple inflator contract that implements theIOpInflator interface. This is the contract that will be called through theBundleBulker contract to decompress the compressed user operation bytes and submit the decompressed user operation through the EntryPoint.

TheIOpInflator interface is defined as follows:

```

File IOpInflator.sol // SPDX-License-Identifier: GPL-3.0-or-later pragmasolidity>=0.8;

import"account-abstraction/interfaces/IEntryPoint.sol";

interfaceIOpInflator{ functioninflate( bytescalldatacompressed )externalviewreturns(UserOperationmemoryop); }

```

For the sake of simplicity, we will write a simple inflator contract that encodes and decodes the user operation bytes using theLZ77 lossless compression algorithm leveragingSolady's solidity implementation .

In practice, you will want to write a more complex inflator contract that implements a more stateful decompression algorithm, likely leveraging important context you as the developer have about the types of user operations you are submitting.

For an example of an efficient, production-ready inflator contract, seeDaimo's implementation which is able to compress their user operation data by over 80%. ```

File SimpleInflator.sol // SPDX-License-Identifier: GPL-3.0-or-later pragmasolidity>=0.8; pragmaabicoderv2;

import"bulk/src/IOpInflator.sol"; import{LibZip}from"solady/src/utils/LibZip.sol";

contractSimpleInflatorisIOpInflator{ functioninflate(bytescalldatacompressed)externalpureoverridereturns(UserOperationmemoryop) { op=abi.decode(LibZip.flzDecompress(compressed), (UserOperation)); }

functioncompress(UserOperationmemoryop)externalpurereturns(bytesmemorycompressed) { compressed=LibZip.flzCompress(abi.encode(op)); } }

```

Thecompress function is not a part of theIOpInflator interface, and is only implemented in theSimpleInflator for simplicity. It is recommended to compress user operations off-chain instead. Once you have written your decompressor contract, you will need to deploy it to the network you are using and verify the source code on Etherscan. We recommend usingforge for this.

```

Terminal Terminal forgecreate\ --rpc-url\ --private-key\ --etherscan-api-key\ --verify\ src/SimpleInflator.sol:SimpleInflator

```

## Get the Inflator contract address whitelisted by Pimlico
```

Once your decompressor contract is deployed and verified, you will need to get the contract address whitelisted by Pimlico's bundler.

This is required because we have to make sure that there are no security issues with your decompressor contract that could lead to the bundler being exploited or made to waste gas.

This is done byreaching out to our team on telegram.

We have deployed the aboveSimpleInflator contract to0x564c7dC50f8293d070F490Fc31fEc3A0A091b9bB on most networks and whitelisted it in case you want to be able to test the compression flow.

Start submitting your compressed user operations

Tip If you are usingpermissionless.js , there is a useful helper function that will allow you to easily submit compressed user operations. See thesendCompressedUserOperation helper function. Once your decompressor contract is deployed and whitelisted, you can start submitting compressed user operations to Pimlico's bundler using thepimlico_sendCompressedUserOperation JSON-RPC method.

Using the aboveSimpleInflator contract and the permissionless.js function as an example, we can compress and submit a user operation as follows:

example.ts SimpleInflatorAbi.ts smartAccountClient.ts pimlicoBundlerClient.ts ```

File example.ts import{ smartAccountClient, simpleSmartAccount, publicClient }from"./smartAccountClient"; import{ pimlicoBundlerClient }from"./pimlicoBundlerClient"; import{ parseAbiItem, encodeFunctionData, getContract }from"viem" import{ SimpleInflatorAbi }from"./SimpleInflatorAbi";

constcallData=awaitsimpleSmartAccount.encodeCallData({ to:"0x0488bEE1Ec682db0F0E74AB52faFdDdEf10Af123", data:encodeFunctionData({ abi: [parseAbiItem('function mint()')] }), value:0n })

// only if using pimlico constgasPrices=awaitpimlicoBundlerClient.getUserOperationGasPrice()

constuserOperation=awaitsmartAccountClient.prepareUserOperationRequest({ userOperation: { callData,// callData is the only required field in the partial user operation maxFeePerGas: gasPrices.fast.maxFeePerGas, maxPriorityFeePerGas: gasPrices.fast.maxPriorityFeePerGas } })

userOperation.signature=awaitsimpleSmartAccount.signUserOperation(userOperation)

// address of the previously deployed inflator contract constinflatorAddress="0xINFLATOR_ADDRESS" constsimpleInflator=getContract({ abi: SimpleInflatorAbi, address: inflatorAddress, client: { public: publicClient } })

// compress the user operation // you can do this off-chain as well constcompressedUserOperation=awaitsimpleInflator.read.compress([userOperation])

constuserOpHash=pimlicoBundlerClient.sendCompressedUserOperation({ compressedUserOperation, inflatorAddress })

```

Congratulations! You have successfully submitted a compressed user operation to Pimlico's bundler.
```