---

title: Forge description: Use Forge to help develop and test contracts with new precompiles on SUAVE

Writing and compiling contracts for SUAVE is similar to Ethereum, and can be done with familiar local tools like Forge, or even IDEs like Remix. Deploying those contracts to SUAVE is done using the same transaction type as Ethereum, and so can also be achieved from those tools without modifying anything other than the RPC endpoint.

SUAVE makes an extended set of precompiles available in any contract.

In order to make interacting with these precompiles easier, we maintain a repository of contracts and libraries called suave-std.

Suapp examples

The easiest way to start creating your own contracts with Forge is to clone the Suapp Examples repo, as it comes with suave-std already set up as a submodule, as well as a series of examples that use both Forge and the Golang SDK.

bash git clone git@github.com:flashbots/suapp-examples.git && cd suapp-examples

bash git submodule init && git submodule update

Encoding and Decoding Transactions

suave-std is intended to implement core functionality that is likely to be used often by many different kinds of Suapps. One such example of this is encoding and decoding transactions from other chains, which you may need to do when matching different transactions in an order flow auction, or turning bundles into (partial) blocks in a block building contract, or weaving together different transactions on different chains for your unique use case.

Having to encode and decode such transactions is a pain, which is exactly the sort of scenario where suave-std can help:

```solidity import "suave-std/Transactions.sol";

contract Example { function example() { Transactions.Legacy memory legacyTxn0 = Transactions.Legacy({ to: address(0x095E7BAea6a6c7c4c2DfeB977eFac326aF552d87), gas: 50000, gasPrice: 10, value: 10, ... });

    // Encode to RLP
    bytes memory rlp = Transactions.encodeRLP(legacyTxn0);

    // Decode from RLP
    Transactions.Legacy memory legacyTxn1 = Transactions.decodeRLP(rlp);
}

} ```

It can also be difficult to test specific precompiles when using Forge. In order to make this possible, suave-std provides a "SUAVE-enabled" test contract, which you can import and use like this:

```solidity // SPDX-License-Identifier: Unlicense pragma solidity ^0.8.13;

import "forge-std/Test.sol"; import "suave-std/Test.sol"; import "suave-std/Suave.sol";

contract TestForge is Test, SuaveEnabled { address[] public addressList = [0xC8df3686b4Afb2BB53e60EAe97EF043FE03Fb829];

function testConfidentialStore() public {
    Suave.DataRecord memory record = Suave.newDataRecord(0, addressList, addressList, "namespace");

    bytes memory value = abi.encode("suave works with forge!");
    Suave.confidentialStore(record.id, "key1", value);

    bytes memory found = Suave.confidentialRetrieve(record.id, "key1");
    assertEq(keccak256(found), keccak256(value));
}

} ```

You will need to have SUAVE running locally in order to be able to run such a test successfully.

## Deploying Contracts

In order to deploy any contract you write in the SUAPP Example repo, you need to either have SUAVE running locally, or an account on Rigil with some rETH from the faucet.

To deploy on Rigil using Forge, you can run the below:

```
bash forge create --rpc-url https://rpc.rigil.suave.flashbots.net --legacy \ --private-key <your_funded_private_key> examples/app-ofa-private/ofa-private.sol:OFAPrivate
```

In order to deploy locally, you can can change the rpc and private key values:

```
bash forge create --rpc-url http://localhost:8545 --legacy \ --private-key 0x91ab9a7e53c220e6210460b65a7a3bb2ca181412a8a7b43ff336b3df1737ce12 \ examples/app-ofa-private/ofa-private.sol:OFAPrivate
```

You should also note, if you look at the other examples, how the Golang SDK is used not only to deploy contracts, but test various kinds of transactions in order to illustrate how each contract is intended to function. Using the Golang SDK can provide for significantly more flexibility than just deploying your contract via Forge.