Background

Before integrating with Uniswap, it may be helpful for newcomers to review the following background information on some important developer web3 concepts, the structure of our examples, and SDK concepts.

Already familiar with web3 development and/or the basics of our SDK and want to get right to the code? Start with our first guide, Getting a Quote!

Providers

Communication with the blockchain is typically done through a provider and local models of smart contracts and their ABIs.

To achieve this, our examples use the ethers.js library. To instantiate a provider you will need a data source. Our examples offer two options:

- **JSON RPC URL**: If you are working directly with the Ethereum mainnet or a local fork, products such as infura offer JSON RPC URLs for a wide variety of chains and testnets. For our examples, we'll only be using the Ethereum mainnet.
- **Wallet Extension**: If you are connecting to a wallet browser extension, these wallets embed a source directly into the Javascript window object as `window.ethereum`. This object surfaces information about the user's wallets and provides the ability to communicate with the connected chain. Importantly for our examples, it can be used with `ethers.js` to construct a provider.

Uniswap's Runnable Examples

Each guide is accompanied and driven by runnable examples using React to provide a basic UI for interacting with the example. Each examples provides relevant options such as running against a local blockchain or connecting to the Ethereum mainnet directly. You also have the option of using a wallet extension which can be connected to either environment.

Inputs and environment settings are configured in each example's `config.ts` and allows for simple setup and configuration.

Developing and Testing

To test your code, we recommend utilizing a local fork of the Ethereum mainnet. To help facilitate easy testing, each example includes a quickstart for running the local chain with a test wallet. To further test, we also recommend using a wallet extension and connecting to the local chain. Finally, each example can be run against the Ethereum mainnet if desired. Full development instructs can be found in the `README.md` of each code example.

Utility Libraries

Each example is concentrated into a single file within the `libs/` folder of the example, with the entry points noted in each guide and README.

To allow the guides to focus on the SDK's core functionality, additional basic building blocks can be found in each example's `libs` folder. The exported functionality from these files is intended to be the minimum needed for each example and not a complete library for production usage. These also include storing core constants such as definitions for tokens, ABI's, and blockchain addresses that can distract from the core concepts. Below are summaries of the helping libraries you will encounter.

Provider Utilities

`provider.ts` wraps the basics of `ethers.js` and connecting to wallet extensions into an abstracted view of a provider, a wallet address, and the ability to send transactions. It also helps abstract the configured environment you wish to run against in your example without making code changes outside of your configuration.

Wallet Utilities

`wallet.ts` offers the ability to query a wallet (whether connected via an extension or defined in code/config) for its balances and other essential information.

Pool Information

`pool.ts` contains the basic querying of pool information when not essential / core to the relevant guide

Display Utilities

`conversion.ts` provides display and light math wrappers to help show human readable prices when dealing with currency amounts (typically stored as raw numbers and the decimal placement separate for precision reasons) in the form of two functions: `fromReadableAmount` and `toReadableAmount`

Notable SDK Structures and Concepts

When working with the SDK it can be helpful to understand some of the design choices and why they are needed. Below you can find a few important concepts.

ABI's

To allow others to interact with a smart contract, each contract exposes an ABI (Application Binary Interface). As these are defined on the blockchain, we must ensure the correct definitions are provided to our Javascript functions. ABI's are provided from various SDK's and imported in as needed. Some examples will define an ABI directly as needed.

CurrencyAmount and JSBI

Cryptocurrency applications often work with very small fractions of tokens. As a result, high precision is very important. To ensure precision is upheld, the `CurrencyAmount` class helps store exact values as fractions and utilizes JSBI for compatibility across the web. To display these amounts nicely to users, additional work is sometimes required.

Currency

The `Currency` class can represent both native currency (ETH) and an ERC20 `Token`. Currencies vary in their relative value, so the `Token` class allows your application to define the number of decimals needed for each currency along with the currency's address, symbol, and name.

Connect to Uniswap

The Uniswap smart contracts exist on the Ethereum blockchain. Use ethers.js or web3.js to connect your website to Ethereum. Users will need a web3-enabled browser. On desktop this means using the MetaMask extension or something similar. On mobile, web3-compatible browsers include Trust Wallet and Coinbase Wallet. See ethereum.org to learn more.

Factory Contract

The Uniswap factory contract can be used to create exchange contracts for any ERC20 token that does not already have one. It also functions as a registry of ERC20 tokens that have been added to the system, and the exchange with which they are associated.

The factory contract can be instantiated using the factory address and ABI:

Factory Address

```
// mainnet
const factory = '0xc0a47dFe034B400B47bDaD5FecDa2621de6c4d95'

// testnets
const ropsten = '0x9c83dCE8CA20E9aAF9D3efc003b2ea62aBC08351'
const rinkeby = '0xf5D915570BC477f9B8D6C0E980aA81757A3AaC36'
const kovan = '0xD3E51Ef092B2845f10401a0159B2B96e8B6c3D30'
const görli = '0x6Ce570d02D73d4c384b46135E87f8C592A8c86dA'
```

Factory Interface

Creating the factory interface in web3 requires the **factory address** and the **factory ABI**:

```
const factoryABI = [
  {
    name: 'NewExchange',
    inputs: [
      { type: 'address', name: 'token', indexed: true },
```

```
    { type: 'address', name: 'exchange', indexed: true },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'initializeFactory',
  outputs: [],
  inputs: [{ type: 'address', name: 'template' }],
  constant: false,
  payable: false,
  type: 'function',
  gas: 35725,
},
{
  name: 'createExchange',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'address', name: 'token' }],
  constant: false,
  payable: false,
  type: 'function',
  gas: 187911,
},
{
  name: 'getExchange',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'address', name: 'token' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 715,
},
{
  name: 'getToken',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'address', name: 'exchange' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 745,
},
{
  name: 'getTokenWithId',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [{ type: 'uint256', name: 'token_id' }],
  constant: true,
```

```
    payable: false,
    type: 'function',
    gas: 736,
  },
  {
    name: 'exchangeTemplate',
    outputs: [{ type: 'address', name: 'out' }],
    inputs: [],
    constant: true,
    payable: false,
    type: 'function',
    gas: 633,
  },
  {
    name: 'tokenCount',
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [],
    constant: true,
    payable: false,
    type: 'function',
    gas: 663,
  },
]

const factoryContract = new web3.eth.Contract(factoryABI, factoryAddress)
```

Exchange Contracts

Get Exchange Address

There is a separate exchange contract for every ERC20 token. The `getExchange` method in the factory contract can be used to find the Ethereum address associated with an ERC20 token address.

```
const exchangeAddress = factoryContract.methods.getExchange(tokenAddress)
```

If the return value is `0x0000000000000000000000000000000000000000` the token does not yet have an exchange.

Exchange Interface

Creating an exchange interface in web3 requires the **exchange address** and the **exchange ABI**:

```
const exchangeABI = [
  {
    name: 'TokenPurchase',
```

```
  inputs: [
    { type: 'address', name: 'buyer', indexed: true },
    { type: 'uint256', name: 'eth_sold', indexed: true },
    { type: 'uint256', name: 'tokens_bought', indexed: true },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'EthPurchase',
  inputs: [
    { type: 'address', name: 'buyer', indexed: true },
    { type: 'uint256', name: 'tokens_sold', indexed: true },
    { type: 'uint256', name: 'eth_bought', indexed: true },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'AddLiquidity',
  inputs: [
    { type: 'address', name: 'provider', indexed: true },
    { type: 'uint256', name: 'eth_amount', indexed: true },
    { type: 'uint256', name: 'token_amount', indexed: true },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'RemoveLiquidity',
  inputs: [
    { type: 'address', name: 'provider', indexed: true },
    { type: 'uint256', name: 'eth_amount', indexed: true },
    { type: 'uint256', name: 'token_amount', indexed: true },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'Transfer',
  inputs: [
    { type: 'address', name: '_from', indexed: true },
    { type: 'address', name: '_to', indexed: true },
    { type: 'uint256', name: '_value', indexed: false },
  ],
  anonymous: false,
```

```
  type: 'event',
},
{
  name: 'Approval',
  inputs: [
    { type: 'address', name: '_owner', indexed: true },
    { type: 'address', name: '_spender', indexed: true },
    { type: 'uint256', name: '_value', indexed: false },
  ],
  anonymous: false,
  type: 'event',
},
{
  name: 'setup',
  outputs: [],
  inputs: [{ type: 'address', name: 'token_addr' }],
  constant: false,
  payable: false,
  type: 'function',
  gas: 175875,
},
{
  name: 'addLiquidity',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'min_liquidity' },
    { type: 'uint256', name: 'max_tokens' },
    { type: 'uint256', name: 'deadline' },
  ],
  constant: false,
  payable: true,
  type: 'function',
  gas: 82605,
},
{
  name: 'removeLiquidity',
  outputs: [
    { type: 'uint256', name: 'out' },
    { type: 'uint256', name: 'out' },
  ],
  inputs: [
    { type: 'uint256', name: 'amount' },
    { type: 'uint256', name: 'min_eth' },
    { type: 'uint256', name: 'min_tokens' },
    { type: 'uint256', name: 'deadline' },
  ],
```

```
  constant: false,
  payable: false,
  type: 'function',
  gas: 116814,
},
{
  name: '__default__',
  outputs: [],
  inputs: [],
  constant: false,
  payable: true,
  type: 'function',
},
{
  name: 'ethToTokenSwapInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'min_tokens' },
    { type: 'uint256', name: 'deadline' },
  ],
  constant: false,
  payable: true,
  type: 'function',
  gas: 12757,
},
{
  name: 'ethToTokenTransferInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'min_tokens' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
  ],
  constant: false,
  payable: true,
  type: 'function',
  gas: 12965,
},
{
  name: 'ethToTokenSwapOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_bought' },
    { type: 'uint256', name: 'deadline' },
  ],
  constant: false,
```

```
  payable: true,
  type: 'function',
  gas: 50455,
},
{
  name: 'ethToTokenTransferOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_bought' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
  ],
  constant: false,
  payable: true,
  type: 'function',
  gas: 50663,
},
{
  name: 'tokenToEthSwapInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_sold' },
    { type: 'uint256', name: 'min_eth' },
    { type: 'uint256', name: 'deadline' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 47503,
},
{
  name: 'tokenToEthTransferInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_sold' },
    { type: 'uint256', name: 'min_eth' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 47712,
},
{
  name: 'tokenToEthSwapOutput',
```

```
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [
      { type: 'uint256', name: 'eth_bought' },
      { type: 'uint256', name: 'max_tokens' },
      { type: 'uint256', name: 'deadline' },
    ],
    constant: false,
    payable: false,
    type: 'function',
    gas: 50175,
  },
  {
    name: 'tokenToEthTransferOutput',
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [
      { type: 'uint256', name: 'eth_bought' },
      { type: 'uint256', name: 'max_tokens' },
      { type: 'uint256', name: 'deadline' },
      { type: 'address', name: 'recipient' },
    ],
    constant: false,
    payable: false,
    type: 'function',
    gas: 50384,
  },
  {
    name: 'tokenToTokenSwapInput',
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [
      { type: 'uint256', name: 'tokens_sold' },
      { type: 'uint256', name: 'min_tokens_bought' },
      { type: 'uint256', name: 'min_eth_bought' },
      { type: 'uint256', name: 'deadline' },
      { type: 'address', name: 'token_addr' },
    ],
    constant: false,
    payable: false,
    type: 'function',
    gas: 51007,
  },
  {
    name: 'tokenToTokenTransferInput',
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [
      { type: 'uint256', name: 'tokens_sold' },
      { type: 'uint256', name: 'min_tokens_bought' },
```

```
    { type: 'uint256', name: 'min_eth_bought' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
    { type: 'address', name: 'token_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 51098,
},
{
  name: 'tokenToTokenSwapOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_bought' },
    { type: 'uint256', name: 'max_tokens_sold' },
    { type: 'uint256', name: 'max_eth_sold' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'token_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 54928,
},
{
  name: 'tokenToTokenTransferOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_bought' },
    { type: 'uint256', name: 'max_tokens_sold' },
    { type: 'uint256', name: 'max_eth_sold' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
    { type: 'address', name: 'token_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 55019,
},
{
  name: 'tokenToExchangeSwapInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_sold' },
```

```
    { type: 'uint256', name: 'min_tokens_bought' },
    { type: 'uint256', name: 'min_eth_bought' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'exchange_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 49342,
},
{
  name: 'tokenToExchangeTransferInput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_sold' },
    { type: 'uint256', name: 'min_tokens_bought' },
    { type: 'uint256', name: 'min_eth_bought' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
    { type: 'address', name: 'exchange_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 49532,
},
{
  name: 'tokenToExchangeSwapOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_bought' },
    { type: 'uint256', name: 'max_tokens_sold' },
    { type: 'uint256', name: 'max_eth_sold' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'exchange_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 53233,
},
{
  name: 'tokenToExchangeTransferOutput',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'uint256', name: 'tokens_bought' },
```

```
    { type: 'uint256', name: 'max_tokens_sold' },
    { type: 'uint256', name: 'max_eth_sold' },
    { type: 'uint256', name: 'deadline' },
    { type: 'address', name: 'recipient' },
    { type: 'address', name: 'exchange_addr' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 53423,
},
{
  name: 'getEthToTokenInputPrice',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [{ type: 'uint256', name: 'eth_sold' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 5542,
},
{
  name: 'getEthToTokenOutputPrice',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [{ type: 'uint256', name: 'tokens_bought' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 6872,
},
{
  name: 'getTokenToEthInputPrice',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [{ type: 'uint256', name: 'tokens_sold' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 5637,
},
{
  name: 'getTokenToEthOutputPrice',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [{ type: 'uint256', name: 'eth_bought' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 6897,
```

```
},
{
  name: 'tokenAddress',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1413,
},
{
  name: 'factoryAddress',
  outputs: [{ type: 'address', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1443,
},
{
  name: 'balanceOf',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [{ type: 'address', name: '_owner' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1645,
},
{
  name: 'transfer',
  outputs: [{ type: 'bool', name: 'out' }],
  inputs: [
    { type: 'address', name: '_to' },
    { type: 'uint256', name: '_value' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 75034,
},
{
  name: 'transferFrom',
  outputs: [{ type: 'bool', name: 'out' }],
  inputs: [
    { type: 'address', name: '_from' },
    { type: 'address', name: '_to' },
```

```
    { type: 'uint256', name: '_value' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 110907,
},
{
  name: 'approve',
  outputs: [{ type: 'bool', name: 'out' }],
  inputs: [
    { type: 'address', name: '_spender' },
    { type: 'uint256', name: '_value' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 38769,
},
{
  name: 'allowance',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'address', name: '_owner' },
    { type: 'address', name: '_spender' },
  ],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1925,
},
{
  name: 'name',
  outputs: [{ type: 'bytes32', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1623,
},
{
  name: 'symbol',
  outputs: [{ type: 'bytes32', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
```

```
    type: 'function',
    gas: 1653,
  },
  {
    name: 'decimals',
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [],
    constant: true,
    payable: false,
    type: 'function',
    gas: 1683,
  },
  {
    name: 'totalSupply',
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [],
    constant: true,
    payable: false,
    type: 'function',
    gas: 1713,
  },
]

const exchangeContract = new web3.eth.Contract(exchangeABI, exchangeAddress)
```

Token Contracts

Some Uniswap interactions require making calls directly to ERC20 token contracts rather than the exchanges with which they are associated.

Get Token Address

The `getToken` method in the factory contract can be used to find the ERC20 token address associated with an exchange contract. There is no barrier of entry for adding an ERC20 token to Uniswap or checks on the validity of the token contracts. Frontend interfaces should maintain a list of valid ERC20 tokens that users can safely trade or allow users to paste in arbitrary addresses.

```
const tokenAddress = factoryContract.methods.getToken(exchangeAddress)
```

If the return value is `0x0000000000000000000000000000000000000000` the input address is not a Uniswap exchange.

Token Interface

Creating a token interface in web3 requires the **token address** and the **token ABI**:

```
const tokenABI = [
  {
    name: 'Transfer',
    inputs: [
      { type: 'address', name: '_from', indexed: true },
      { type: 'address', name: '_to', indexed: true },
      { type: 'uint256', name: '_value', indexed: false },
    ],
    anonymous: false,
    type: 'event',
  },
  {
    name: 'Approval',
    inputs: [
      { type: 'address', name: '_owner', indexed: true },
      { type: 'address', name: '_spender', indexed: true },
      { type: 'uint256', name: '_value', indexed: false },
    ],
    anonymous: false,
    type: 'event',
  },
  {
    name: '__init__',
    outputs: [],
    inputs: [
      { type: 'bytes32', name: '_name' },
      { type: 'bytes32', name: '_symbol' },
      { type: 'uint256', name: '_decimals' },
      { type: 'uint256', name: '_supply' },
    ],
    constant: false,
    payable: false,
    type: 'constructor',
  },
  {
    name: 'deposit',
    outputs: [],
    inputs: [],
    constant: false,
    payable: true,
    type: 'function',
    gas: 74279,
  },
  {
    name: 'withdraw',
    outputs: [{ type: 'bool', name: 'out' }],
```

```
  inputs: [{ type: 'uint256', name: '_value' }],
  constant: false,
  payable: false,
  type: 'function',
  gas: 108706,
},
{
  name: 'totalSupply',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 543,
},
{
  name: 'balanceOf',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [{ type: 'address', name: '_owner' }],
  constant: true,
  payable: false,
  type: 'function',
  gas: 745,
},
{
  name: 'transfer',
  outputs: [{ type: 'bool', name: 'out' }],
  inputs: [
    { type: 'address', name: '_to' },
    { type: 'uint256', name: '_value' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 74698,
},
{
  name: 'transferFrom',
  outputs: [{ type: 'bool', name: 'out' }],
  inputs: [
    { type: 'address', name: '_from' },
    { type: 'address', name: '_to' },
    { type: 'uint256', name: '_value' },
  ],
  constant: false,
  payable: false,
```

```
  type: 'function',
  gas: 110600,
},
{
  name: 'approve',
  outputs: [{ type: 'bool', name: 'out' }],
  inputs: [
    { type: 'address', name: '_spender' },
    { type: 'uint256', name: '_value' },
  ],
  constant: false,
  payable: false,
  type: 'function',
  gas: 37888,
},
{
  name: 'allowance',
  outputs: [{ type: 'uint256', name: 'out' }],
  inputs: [
    { type: 'address', name: '_owner' },
    { type: 'address', name: '_spender' },
  ],
  constant: true,
  payable: false,
  type: 'function',
  gas: 1025,
},
{
  name: 'name',
  outputs: [{ type: 'bytes32', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 723,
},
{
  name: 'symbol',
  outputs: [{ type: 'bytes32', name: 'out' }],
  inputs: [],
  constant: true,
  payable: false,
  type: 'function',
  gas: 753,
},
{
```

```
    name: 'decimals',
    outputs: [{ type: 'uint256', name: 'out' }],
    inputs: [],
    constant: true,
    payable: false,
    type: 'function',
    gas: 783,
  },
]

const tokenContract = new web3.eth.Contract(tokenABI, tokenAddress)
```

Connecting to Wallets

Introduction

This guide will cover how to connect wallets with `web3-react`. It is based on the `web3-react` example found in the Uniswap code examples repository. To run this example, check out the examples's README and follow the setup instructions.

In this example we will walk through setting up `web3-react` and connecting the most popular browser-injected connector, MetaMask, using [@web3-react/metamask](https://www.npmjs.com/package/@web3-react/metamask).

The input parameters to this guide are the chains that we want our app to be able to connect to and their RPC URLs.

The guide will **cover**:

1. Creating a `web3-react` `Web3ReactProvider`
2. Building a `web3-react` `InjectedConnector`
3. Connecting and disconnecting the application to the connector

At the end of the guide, we should be able to connect and disconnect your dApp to a MetaMask connector.

For this guide, the following `web3-react` packages are used:

- `@web3-react/core`
- `@web3-react/types`
- `@web3-react/metamask`

This guide uses `web3-react` version 8, which is a beta version.

These will be automatically installed by following the example's README.

The core code of this guide can be found in Web3ContextProvider and InjectedConnector.

Creating a `Web3ReactProvider`

To be able to interact with the methods that `web3-react` offers, we first need to setup a `Web3ReactProvider` and wrap our application in it. `web3-react`

uses a React `Context` to allow us to use the exposed hooks without additional configuration.

To start, we create a React component called `Web3ContextProvider` in order to wrap the logic of configuring the `Web3ReactProvider`. In this component, we first import `Web3ReactProvider` from [@web3-react/core](https://www.npmjs.com/package/@web3-react/core).

The component receives just one prop which is the `children` to which it will be providing the `Context`:

```
typescript reference title="Defining the Web3React component"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/81
```

We then implement the component by rendering the imported `Web3ReactProvider` with the `children` within that:

```
typescript reference title="Implementing the component" referenceLinkText="View
on Github" customStyling https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5
```

Note that we map our list of `Connections` to a *tuple* of the `connector` and `hooks` of the connection. The third element of a `Connection` refers to the type of `Connection` being established, which we will later use to keep track of the actively connected wallet.

Finally, having created the `Web3ContextProvider` component, we can navigate to our index file and wrap the whole application with it:

```
typescript reference title="Wrapping our app with the web3 context"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/7a
```

Building an Injected Connector

The only parameter that we provided to the `Web3ReactProvider` component is a list of prioritized connectors, declared as `PRIORITIZED_CONNECTORS`. The prioritization ordering is with regards to which connector we want to be active when more than one connector is connected to our application. The list is defined inside our connectors module: `typescript reference title="Creating the prioritized Connectors list" referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af944094191f39da62a9`

Each one of those connectors lives within its own file, and they all follow a similar setup pattern.

An example of a connector in the list is the `InjectedConnector`, which supports wallets that inject an *Ethereum Provider* into the browser window. The most popular example of an injected connector is the *MetaMask* browser extension. To set it up, we import `initializeConnector` function from core and the `MetaMask` type from metamask:

```
typescript reference title="Importing Connector dependencies"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/85
```

We then utilize the templated `initializeConnector` function with `MetaMask` as the type argument:

```
typescript reference title="Initializing the MetaMask connector"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/81
```

By passing in `MetaMask` as the type argument, we define the function's required input parameters. In this case, the only parameter we need to pass is an instance of `Metamask`, which receives the `actions` and `onError` parameters. The first parameter defines the actions that `web3-react` performs on its local store for the connector (this usually can be passed through without modification), while the second parameter is the callback invoked when an error occurs.

The return type of the function is a tuple of the initialized `Connector` and the `Hooks` that we can use on it. Using this tuple, we create an instance of a `Connection` type, by setting the `type` property to `INJECTED`:

```
typescript reference title="Creating a connection instance"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/85
```

Finally, we return the instance of `Connection` we created, which is added to the list of prioritized connectors.

For help on creating the rest of the supported connectors of this examples, please visit our connectors page!

Connecting and disconnecting the application to the connector

Having built our `InjectedConnector`, we can now use it in the Context that allows our application to use that connector:

```
typescript reference title="Creating the Option component" referenceLinkText="View
on Github" customStyling https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5
```

The component receives 5 parameters: - `isEnabled` determines whether the connector is eligible to be activated - `isConnected` determines whether the connector is currently active - `connectionType` determines the `ConnectionType` - `onActivate` is called once the component has established a connection - `onDeactivate` is called once the component has disconnected

In the case of *MetaMask*, when declaring the `InjectedConnector` we pass the connector-specific arguments:

```
typescript reference title="Creating an injected connector"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/81
```

Then, in the `html` portion of the `Option`, we can figure out whether we want the current `Option`'s action button to be disabled, and whether clicking the button would result in the connector being connected or disconnected:

```
typescript reference title="The component user interface" referenceLinkText="View
on Github" customStyling https://github.com/Uniswap/examples/blob/81ec93e97b0afded621e177fe5
```

Finally, we also have enough information to figure out what action to take when the button is clicked. In the case that the click triggers a connection:

```
typescript reference title="On connecting to a Connector" referenceLinkText="View
on Github" customStyling https://github.com/Uniswap/examples/blob/8c0e36ca8d2ba4718af9440941
```

To connect our wallet, all we need to do is to call the `tryActivateConnector` function and pass it the `InjectedConnector`. We then call the `onActivate` callback, which makes the `InjectedConnector` the active connector in our application's state.

`tryActivateConnector` takes as its argument the connector that we want to activate, and attempts to call `activate` on it. If this activation succeeds, it returns the new `ConnectionType`:

```
typescript reference title="The implementation of tryActivateConnector"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/8c
```

In the case that the click triggers a disconnection:

```
typescript reference title="On disconnecting from a Connector"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/8c
```

To disconnect, all we need to do is to call `tryDeactivateConnector` and pass in it the `InjectedConnector` we created before. We then call the `onDeactivate` callback, which removes the `InjectedConnector` as the currently active connector from our application's state.

`tryDeactivateConnector` takes as its argument the connector that we want to deactivate, and attempts to call `deactivate` on it. If this deactivation succeeds, it resets the connector's state by calling `resetState` and returns `null`:

```
typescript reference title="The implementation of tryDeactivateConnector"
referenceLinkText="View on Github" customStyling https://github.com/Uniswap/examples/blob/81
```

Next Steps

Now that we have gone through connecting and disconnecting from an `InjectedConnector`, we will learn how to connect and disconnect from all the different types of connectors that `web3-react` supports.

Factory

Factory

Code

`UniswapV2Factory.sol`

Address

`UniswapV2Factory` is deployed at `0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f`
on the Ethereum mainnet, and the Ropsten, Rinkeby, Görli, and Kovan testnets.
It was built from commit 8160750.

Events

PairCreated

```
event PairCreated(address indexed token0, address indexed token1, address pair, uint);
```

Emitted each time a pair is created via createPair.

- `token0` is guaranteed to be strictly less than `token1` by sort order.
- The final `uint` log value will be `1` for the first pair created, `2` for the second, etc. (see allPairs/getPair).

Read-Only Functions

getPair

```
function getPair(address tokenA, address tokenB) external view returns (address pair);
```

Returns the address of the pair for `tokenA` and `tokenB`, if it has been created,
else `address(0)` (`0x0000000000000000000000000000000000000000`).

- `tokenA` and `tokenB` are interchangeable.
- Pair addresses can also be calculated deterministically via the SDK.

allPairs

`function allPairs(uint) external view returns (address pair);`

Returns the address of the `n`th pair (`0`-indexed) created through the factory, or `address(0)` (`0x0000000000000000000000000000000000000000`) if not enough pairs have been created yet.

- Pass `0` for the address of the first pair created, `1` for the second, etc.

allPairsLength

`function allPairsLength() external view returns (uint);`

Returns the total number of pairs created through the factory so far.

feeTo

`function feeTo() external view returns (address);`

See Protocol Charge Calculation.

feeToSetter

`function feeToSetter() external view returns (address);`

The address allowed to change feeTo.

State-Changing Functions

createPair

`function createPair(address tokenA, address tokenB) external returns (address pair);`

Creates a pair for `tokenA` and `tokenB` if one doesn't exist already.

- `tokenA` and `tokenB` are interchangeable.
- Emits PairCreated.

Interface

```
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol';

pragma solidity >=0.5.0;

interface IUniswapV2Factory {
  event PairCreated(address indexed token0, address indexed token1, address pair, uint);

  function getPair(address tokenA, address tokenB) external view returns (address pair);
  function allPairs(uint) external view returns (address pair);
```

```
  function allPairsLength() external view returns (uint);

  function feeTo() external view returns (address);
  function feeToSetter() external view returns (address);

  function createPair(address tokenA, address tokenB) external returns (address pair);
}
```

ABI

```
import IUniswapV2Factory from '@uniswap/v2-core/build/IUniswapV2Factory.json'
```

https://unpkg.com/@uniswap/v2-core@1.0.0/build/IUniswapV2Factory.json

Fees

Liquidity provider fees

There is a **0.3%** fee for swapping tokens. **This fee is split by liquidity providers proportional to their contribution to liquidity reserves.**

Swapping fees are immediately deposited into liquidity reserves. This increases the value of liquidity tokens, functioning as a payout to all liquidity providers proportional to their share of the pool. Fees are collected by burning liquidity tokens to remove a proportional share of the underlying reserves.

Since fees are added to liquidity pools, the invariant increases at the end of every trade. Within a single transaction, the invariant represents `token0_pool / token1_pool` at the end of the previous transaction.

There are many community-developed tools to determine returns. You can also read more in the docs about how to think about LP returns.

Protocol Fees

At the moment there are no protocol fees. However, it is possible for a 0.05% fee to be turned on in the future.

More information about a potential future protocol fee can be found here.

Protocol Charge Calculation

In the future, it is possible that a protocol-wide charge of 0.05% per trade will take effect. This represents th (16.6%) of the 0.30% fee. The fee is in effect if feeTo is not `address(0)` (`0x0000000000000000000000000000000000000000`), indicating that feeTo is the recipient of the charge.

This amount would not affect the fee paid by traders, but would affect the amount received by liquidity providers.

Rather than calculating this charge on swaps, which would significantly increase gas costs for all users, the charge is instead calculated when liquidity is added or removed. See the whitepaper for more details.

Getting Started

The pages that follow contain technical reference information on the Uniswap SDK. Looking for a quick start instead? You may also want to jump into a guide, which offers a friendlier introduction to the SDK!

The SDK is written in TypeScript, has a robust test suite, performs arbitrary precision arithmetic, and supports rounding to significant digits or fixed decimal places. The principal exports of the SDK are *entities*: classes that contain initialization and validation checks, necessary data fields, and helper functions.

An important concept in the SDK is *fractions*. Because Solidity performs integer math, care must be taken in non-EVM environments to faithfully replicate the actual computation carried out on-chain. The first concern here is to ensure that an overflow-safe integer implementation is used. Ideally, the SDK would be able to use native BigInts. However, until support becomes more widespread, JSBI objects are used instead, with the idea that once BigInts proliferate, this dependency can be compiled away. The second concern is precision loss due to, for example, chained price ratio calculations. To address this issue, all math operations are performed as fraction operations, ensuring arbitrary precision up until the point that values are rounded for display purposes, or truncated to fit inside a fixed bit width.

The SDK works for all chains on which the factory is deployed.

Code

The source code is available on GitHub.

Dependencies

The SDK declares its dependencies as peer dependencies. This is for two reasons:

- prevent installation of unused dependencies (e.g. `@ethersproject/providers` and `@ethersproject/contracts`, only used in `Fetcher`)
- prevent duplicate `@ethersproject` dependencies with conflicting versions

However, this means you must install these dependencies alongside the SDK, if you do not already have them installed.