# Liquality Integration Guide

This is a quick guide that describes all steps that are needed when a new chain is integrated in liquality ecosystem. In the last section will be described some chain specific issues that are important for all developers using this guide.

## Step I. Integration in Chain Abstraction Layer

Chain abstraction layer (CAL) is a monorepo that holds sdk like modules that expose a common interface and a bridge between Wallet and Atomic Agent and the blockchain itself. Here is the link to CAL repository:
https://github.com/liquality/chainabstractionlayer

Every new blockchain should implement its packages in chain abstraction layer in packages folder: https://github.com/liquality/chainabstractionlayer/tree/dev/packages

- [Chain-Name]-js-wallet-provider - holds key management and signing
- [Chain-Name]-network - holds different networks for a given chain e.g testnet, mainnet, etc.
- [Chain-Name]-rpc-provider - logic for rpc connection with the given blockchain
-
- [Chain-Name]-near-swap-find-provider - it is used to fetch data about the given account and its transaction and swaps
- [Chain-Name]-swap-provider - implements main logic for create/withdraw/refund swap/order using HTLC contracts
- [Chain-Name]-utils - a helper module that holds some utility functions in order to support the other modules
- [Chain-Name]-rpc-fee-provider - a module that computes and returns current gas price needed for the transaction

This is the bare minimum that is required in order for a new chain to be integrated in liquality ecosystem (there are some exceptions if we are integrating evm compatible blockchain similar to Ethereum)

## Step II. Integration in Cryptoassets

Liquality cryptoassets is a package that stores general information for all supported blockchain in the ecosystem, their symbols, and decimals.
https://github.com/liquality/cryptoassets/

New chains should be imported in the json file stored here:
https://github.com/liquality/cryptoassets/blob/master/src/assets/network.js

## Step III. Integration in Wallet

Liquality wallet is a browser extension that is written in vuejs. It is super useful and easy to use and anyone can download it freely on chrome web store.
https://github.com/liquality/wallet

Adding new token in wallet is relatively simple:

- Add an svg for the new chain here: src/assets/icons/assets/[chain-name].svg
- Add the symbol of the new chain here: src/build.config.js
- Add a provider of the new chain here: src/contentScript.js
- Add new provider following the logic for the other chains here: src/inject.js
- Add all implemented providers for the given chain in step 1 (Chain Abstraction Layer) in the factory: src/store/factory/client.js
- Add the Symbol of the new chain in the getters: src/store/getters.js
- Add all supported networks for the given chain:  src/utils/asset.js
- Add some fixed fee values if available: src/utils/fees.js
- Add faucet url if available: src/views/Receive.vue

## Step IV. Integration in Atomic Agent

Atomic Agent is a piece of code that works as an automated market-making software. When someone is using the Wallet extension he/she is swapping/trading agents Atomic Agent instances.
https://github.com/liquality/atomicagent

These are the files that should be changed for the integration:

- Add the Symbol and decimals of the new chain src/migrate/data/assets.json
- Add new trading pairs including the new coin/token and put initial rates here: src/migrate/data/markets.json
- Add all providers for the given chain from Chain Abstraction Layer: src/utils/clients.js
- Add example configuration in ./config.toml file

## Step V. Integration in Ethereum Scraper

Ethereum scrapper is a service that fetches all blocks for a given blockchain and stores this data in a database in order to give to atomic agent and wallet quick access.
https://github.com/liquality/ethereum-scraper

Some chains require such kind of scrapper but others do not.
If the chain is Ethereum like the only change that should be done is inside https://github.com/liquality/ethereum-scraper .env file

## VI. Blockchain Specific

### VI.1 Bitcoin Specifics

- Bitcoin transaction fees can be computed in deterministic manner and they are subtracted from the transaction value
- Uses Bitcoin Script as a simple smart contract language (Non-turing-complete)
- Blocktime ~ 10 minutes, Decimals: 8

### VI.2 Ethereum Specifics

- Ethereum transactions fees are dynamic and should be updated on every transaction
- Uses solidity as a smart contract language

- Blocktime ~ 13 sec, Decimals: 18

### VI.3 Near Specifics

- Near accounts should be funded in order to exist on the blockchain
- Near transaction fees can be computed in deterministic manner and they are subtracted from the transaction value similar to Bitcoin
- Uses Asembly Script as a smart contract language
- Blocktime: ~ 1s, Decimals: 24

### VI.4 Ethereum like chains specifics

- Ethereum like chains e.g RSK, BSC can use all Ethereum providers from CAL without providing their own