Water Layer: Omnichain Inscription Standard with Runtime

contact@waterlayer.xyz
https://twitter.com/waterlayer_xyz

Abstract

Water Layer introduces a set of inscription-based assets and execution protocols (i.e. Water Standard), with a runtime (i.e. Water Client) to support different standards of inscription assets and flexible execution across different blockchains. Water Layer aims to unlock a decentralized, permissionless, scalable and interoperable blockchain paradigm.

Artifacts

- **Water Standard**: the inscription standards for Water Layer
  - Water Standard Asset: defines asset in any blockchain, natively interoperable
    - Fungible token: WRC20, and etc.
    - Non-fungible token: WRC721, and etc.
  - Water Standard Execution (WSE): defines client-side execution logic within different VMs (e.g.Solidity for EVM).
- **Water Client**: any client that implements Water Standard, consisting of indexers and runtimes, which can be implemented in any language.

Value Proposition

1. Omnichain inscription: support all major existing asset inscription standards across different blockchains including Etherscription, BRC20, ARC20, etc. via Water Client.
2. Permissionless: permissionless for anyone to set up a local client with a RPC endpoint or a blockchain node to interact with Water Layer or adopt a 3rd-party client provider (e.g. Unisat, OKX Wallet).

3. Interoperability: any WRC20 assets minted on any blockchain can be bridged to other blockchains in a trustless way. Cross-chain transfer method is natively defined as a method of WRC20 assets.
4. Scalability: computation is much more cost-efficient compared to blockchain with client-side execution within Water Client. States are indexed and computed off-chain and only inscriptions are stored on-chain, which eliminates the cost for on-chain state storage, compute, and synchronization.

Water Standard

Asset Protocol (WRC20)

method	***arguments***	***example***

deploy	• p (required): protocol name
	• op (required): event type
	• tick (required): ticker of the asset, max 5 letters
	• max (optional): max supply
	• lim (optional): mint limit, default to 10000
	• dec (optional): decimal precision, default to 18
		{
		"p": "wrc-20",
		"op": "deploy",
		"tick": "wrc",
		"max": "21000000",
		"lim": "1000"
		}

mint	• p (required): protocol name
	• op (required): event type
	• tick (required): ticker of the asset, max 5 letters
	• amt (required): the amount of the tick asset to mint. Has to be smaller than "lim" in deploy
		{
		"p": "wrc-20",
		"op": "mint",
		"tick": "wrc",
		"amt": "1000"
		}

transfer	• p (required): protocol name
	• op (required): event type
	• tick (required): ticker of the asset, max 5 letters
	• amt (required): the amount of the tick asset to mint. Has to be smaller than "lim" in deploy
		{
		"p": "wrc-20",
		"op": "transfer",
		"tick": "wrc",
		"amt": "100"
		}

teleport	• p (required): protocol name
	• op (required): event type
	• tick (required): ticker of the asset, max 5 letters
	• amt (required): the amount of the tick asset to mint. Has to be smaller than "lim" in deploy
	• tc (required): target chain name
	• ta (required): address on target chain
		{
		"p": "wrc-20",
		"op": "teleport",
		"tick": "wrc",
		"amt": "10000",
		"tc": "solana",
		"ta": "7rhxnLVs23s"
		}

Notice, deploy, mint, and transfer are compatible with BRC20, with teleport being the new method in WRC20 standard, enabling cross-chain interoperability for Inscription-based assets.

Execution Protocol (WSE)

method	***arguments***	***example***

deploy	• p (required): protocol name
	• op (required): event type
	• r (required): runtime name
	• rv (required): runtime version
	• b (required): bytecode of the executable
	• i (required): interface of the executable
		{
		"p": "wse",
		"op": "deploy",
		"r": "evm",
		"rv": "13.4",
		"b": "3ca21bb9e2776ce2”,
		"i": "{ "get_value": "...",
		"set_value": "..."
		}"
		}

call	• p (required): protocol name
	• v (required): protocol version
	• op (required): event type
	• c (required): chain name
	• hash (required): tx hash of the “deploy” inscription transaction
	• m (required): method name
	• arg0 (optional): argument0 to call
	• arg1 (optional): argument1 to call
	• …
		{
		"p": "wse",
		"op": "call",
		"c": "ethereum",
		"hash": "cc397eb2c3aab3",
		"m": "set_value",
		"arg0": "10"
		}

Water Client

Water Client is responsible to offchain index the inscriptions, and perform client-side execution to update the states and provide an interface for end users to interact with. A wallet supporting Water Standard needs to be backed by a Water Client to get the most up-to-date states.

Water Client can have multiple runtimes executing inscriptions from WSE (check out r and rv arguments in deploy).

Roadmap

Dec 2023: WRC20 token minting with limited runtime support (i.e. EVM)
Jan 2024: WRC20 asset swap (i.e. limited WSE functionality)
Mar 2024: Full WSE implementation
Q2 2024+: Support wrapping inscription assets w/ different standards into WRC20, and more runtimes