Frequency

Your Protocol, Your Rules

Build and Launch a Custom Data Protocol on Frequency

Web3 Summit 2025

What is Frequency?

What is Frequency?

Abstract User Identity

Service Delegation

Custom Data Protocols

Stake-based Economics

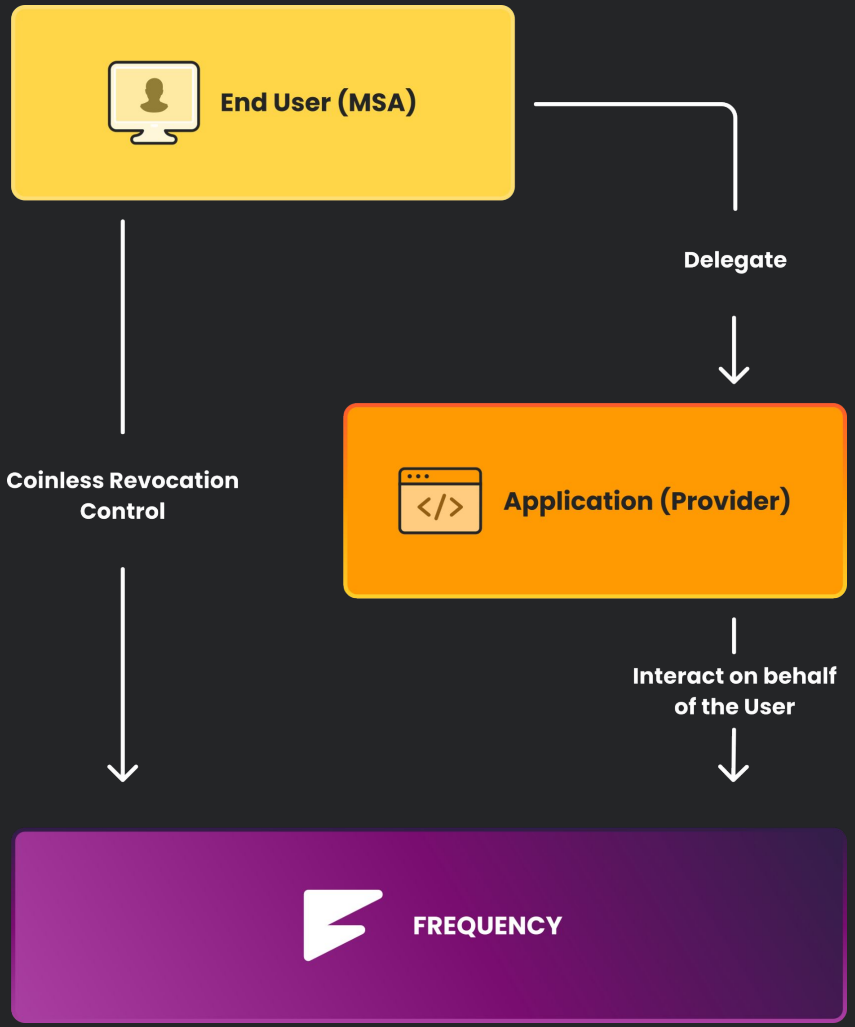
Details

Abstract User Identity

- Abstract accounts (MSAs) with a 64-bit Id
- User's wallet provides the locus of control
- Optional Account Data: handle, permissions, etc...
- Usually coinless, even for direct revocation actions
- Supports receiving tokens

Service Delegation

- Users authorize
  - One-time
  - Ongoing Delegation
- Providers interact

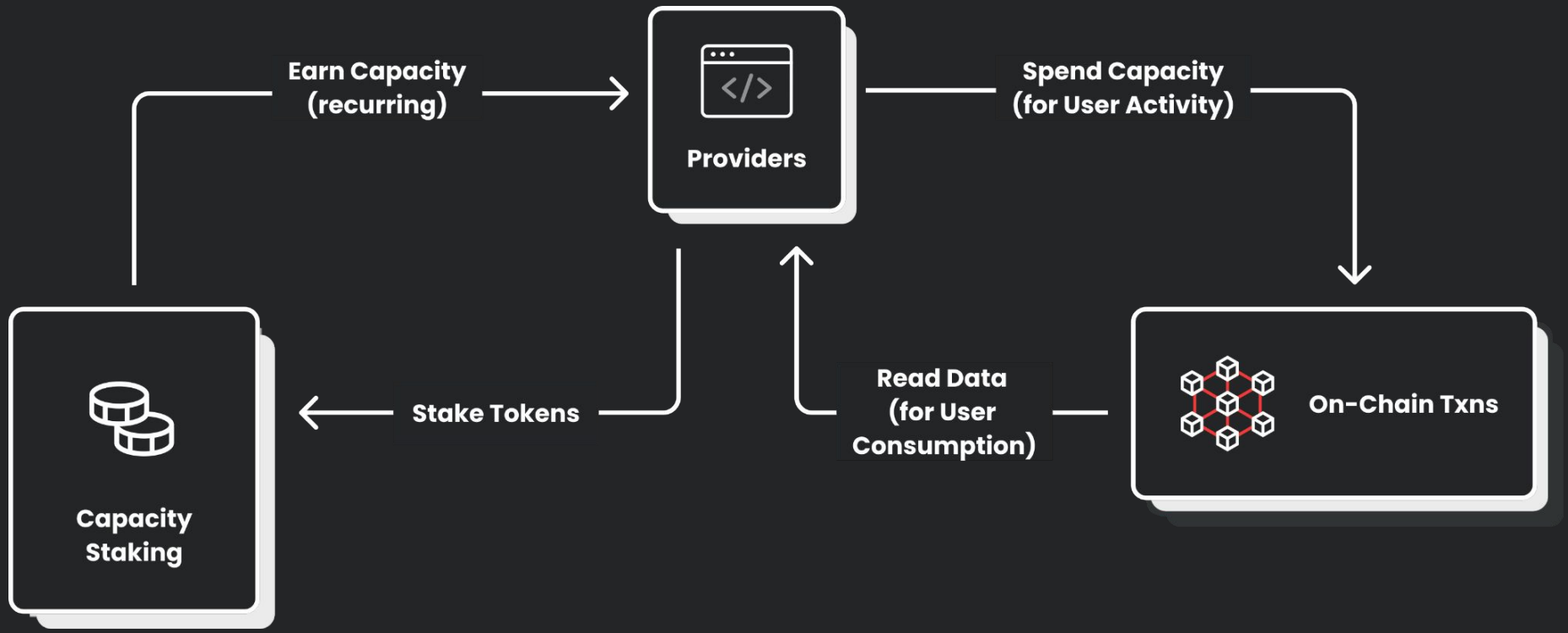

End User (MSA)
Delegate
Coinless Revocation Control
Application (Provider)
Interact on behalf of the User
FREQUENCY

Custom Data Protocols

- *Discovery* method
- *Storage* type
- *Structure* definition

Stake-based Economics

- *Capacity* offers renewable, rate-limited access to perform any Capacity-enabled transaction.
- Predictable costs allow companies to provide services to their customers
- Shared data is shared value. The value of data increases as the amount of data increases.


Earn Capacity
(recurring)
Providers
Spend Capacity
(for User Activity)
Capacity
Staking
Stake Tokens
Read Data
(for User
Consumption)
On-Chain Txns

What is Frequency?

Abstract User Identity

On-chain account system controlled by the user

Service Delegation

Users can delegate actions to service Providers

Custom Data Protocols

Protocols are defined via schemas and storage options

Stake-based Economics

Refillable Capacity allows Providers to service users

Goal: Build, launch, and use a custom data protocol on Frequency Testnet

Custom Data Protocols

Frequency Data Questions

- *Discovery*: How do I want this data accessed?
- *Storage*: Where is the data stored?
- *Schema*: How is this data structured?

Frequency Data Discovery Options

Frequency Data Discovery Options

- Account Data
  - Specific data about the account
  - Example: Handles
  - Chain-defined

Frequency Data Discovery Options

- ~~Account Data~~
- User-Centric Data
  - I want to discover something about this user
  - Example: Social Graph
  - Schema-defined
- Time-Centric Data
  - I want to know what happened at a point in time
  - Example: Content References
  - Schema-defined

Frequency Data Storage Options

Frequency Data Storage Options

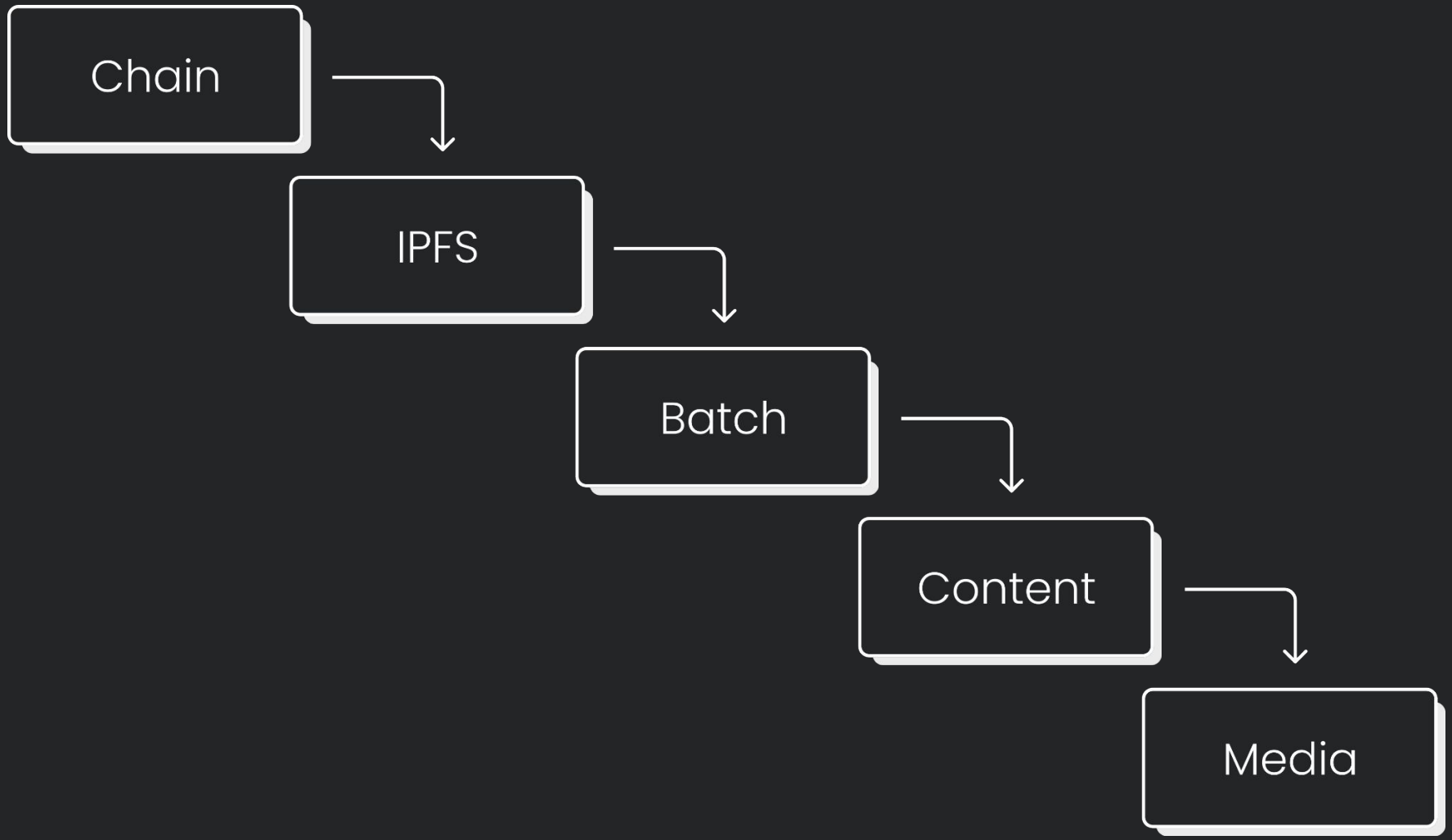
- On-Chain
- Off-Chain

On-Chain Data Storage

- Very Limited
  - Encrypted Data
  - Public Key and Address Data
  - Relationship Data
- Generally Computer-Driven

Off-Chain Data Storage

- Chain References: IPFS
- Secondary References: Specification-Defined
- Batching
  - Stream of Messages
  - Pointers and References
- Wide-Open, User-Driven Data


Chain
IPFS
Batch
Content
Media

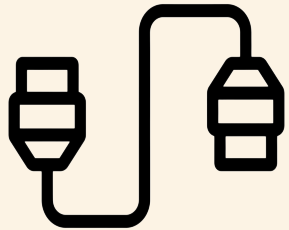
Frequency Data Schemas

Frequency Data Schemas

- Every message on Frequency has a Schema
- Schemas answer three questions:

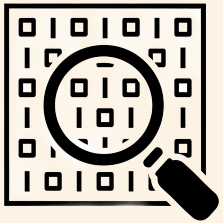
Frequency Data Schemas

- Every message on Frequency has a Schema
- Schemas answer three questions:
  - Meaning: How does this data connect to other data?



Frequency Data Schemas

- Every message on Frequency has a Schema
- Schemas answer three questions:
  - Meaning: How does this data connect to other data?
  - Structure: How can I deserialize this data?



Frequency Data Schemas

- Every message on Frequency has a Schema
- Schemas answer three questions:
  - Meaning: How does this data connect to other data?
  - Structure: How can I deserialize this data?
  - Specification: What are the rules for this data?

Frequency Data Schemas

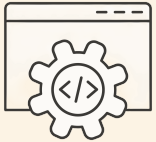
- Every message on Frequency has a Schema
- Schemas answer three questions:
  - Meaning: How does this data connect to other data?
  - Structure: How can I deserialize this data?
  - Specification: What are the rules for this data?
- Permissions are connected to Schemas
  - Signature-Based Permission
  - Delegation-Based Permission

Frequency Data Schemas

- Every message on Frequency has a Schema
- Schemas answer three questions:
  - Meaning: How does this data connect to other data?
  - Structure: How can I deserialize this data?
  - Specification: What are the rules for this data?
- Permissions are connected to Schemas
  - Signature-Based Permission
  - Delegation-Based Permission
- Other Settings & Options
  - Discovery: How do I want this data accessed?
  - Storage: Where do I retrieve this data?

Frequency data flows follow standard patterns

User:
Create Data



Application:
Process Data

Frequency:
Distribute Data

Others:
Consume Data

Examples Time!

Pieces to Answer

- Schema
  - Meaning
  - Structure
  - Specification
- Permissions
- Discovery
- Storage

Token Addresses Protocol

- Schema
  - Meaning: Addresses the user controls on other chains
  - Structure: Avro
  - Specification: New, Uses SLIP-0044
- Permissions: One-time Signature Delegation
- Discovery: User-Indexed
- Storage: On-chain, Itemized

Token Addresses Protocol

```
{
  type: "record",
  name: "DefaultTokenAddress",
  namespace: "frequency",
  fields: [
    {
      name: "token_slip_0044",
      type: "int",
      doc: "Network for this token address using SLIP-0044 registered coin type integers",
    },
    {
      name: "address",
      type: "string",
      doc: "The address as a string encoded in standard way for the given coin type",
    },
  ],
}
```

Tangent: Why not just use a smart contract?

DSNP (Decentralized Social Media Protocol)

- Frequency was based on DSNP
- Has multiple, multi-layered Schemas
- Social Graph?
  - On-chain, Stateful Storage (Paginated & Itemized)
- Content?
  - Off-chain, IPFS, Parquet Batches

Public Follows

```
{
  type: "record",
  name: "UserPublicFollowsChunk",
  namespace: "org.dsnp",
  fields: [{ name: "compressedPublicGraph", type: "bytes" }],
  types: [
    {
      type: "array",
      name: "PublicGraph",
      namespace: "org.dsnp",
      items: {
        namespace: "org.dsnp",
        name: "GraphEdge",
        type: "record",
        doc: "A relationship to another DSNP user",
        fields: [
          {
            name: "userId",
            type: "long",
            doc: "The other user's DSNP User Id",
          },
          {
            name: "since",
            type: "long",
            doc: "Timestamp in Unix epoch seconds when this relationship was originally established",
          },
        ],
      },
    },
  ],
}
```

Private Connections

```
{
  type: "record",
  name: "UserPrivateConnectionsChunk",
  namespace: "org.dsnp",
  fields: [
    { name: "keyId", type: "long", doc: "User-Assigned Key Identifier" },
    {
      name: "pridList",
      type: {
        type: "array",
        items: {
          namespace: "org.dsnp",
          name: "PRId",
          type: "fixed",
          size: 8,
          doc: "Pseudonymous Relationship Identifier",
        },
      },
    },
    {
      doc: "lib_sodium sealed box",
      name: "encryptedCompressedPrivateGraph",
      type: "bytes",
    },
  ],
  types: [
    {
      type: "array",
      name: "PrivateGraph",
      namespace: "org.dsnp",
      items: {
        namespace: "org.dsnp",
        name: "GraphEdge",
        type: "record",
        doc: "A relationship to another DSNP user",
        fields: [
          {
```

DSNP Public Content

```
[
  {
    name: "announcementType",
    column_type: { INTEGER: { bit_width: 32, sign: true } },
    compression: "GZIP",
    bloom_filter: false,
  },
  {
    name: "contentHash",
    column_type: "STRING",
    compression: "GZIP",
    bloom_filter: true,
  },
  {
    name: "fromId",
    column_type: { INTEGER: { bit_width: 64, sign: false } },
    compression: "GZIP",
    bloom_filter: true,
  },
  {
    name: "url",
    column_type: "STRING",
    compression: "GZIP",
    bloom_filter: false,
  },
]
```

AT Protocol

- Aka BlueSky
- Publishing an independent “Firehose”
- Offers Historical Replay
- Currently on Testnet
- Three Schemas
  - Account
  - Identity
  - Commits

AT Protocol Example Identity Schema

```
{
  repoDid: { type: 'UTF8', compression: 'GZIP', statistics: false },
  time: { type: 'UINT_64', compression: 'GZIP', statistics: false },
  handle: {
    type: 'UTF8',
    compression: 'GZIP',
    optional: true,
    statistics: false,
  },
  keys: { type: 'UTF8', compression: 'GZIP', statistics: false },
  pdsUrl: { type: 'UTF8', compression: 'GZIP', statistics: false },
  alsoKnownAs: { type: 'UTF8', compression: 'GZIP', statistics: false },
}
```

Build Together Time!

OpenTimestamps

- https://opentimestamps.org
- *“A timestamp proves that a message existed prior to some point in time; timestamps are occasionally referred to as ‘proofs-of-existence’. Being able to prove that data existed prior to a point in time is surprisingly useful.”*
- Deployed right now on Bitcoin

OpenTimestamps: Cost for Three Calendars

- Average time between transactions in the last week:
  - 2.47 hours
  - 4.00 hours
  - 8.84 hours
- Fees used in the last week:
  - `0.00015240` BTC
  - `0.00007334` BTC
  - `0.00007202` BTC
  - Weekly Total: `0.00029776` BTC / `$35.07` USD
  - Annualized on-chain costs: `$1,823.64` USD

OpenTimestamps: Cost for Three Calendars

- Average time between transactions in the last week:
  - 1 hour
  - ~8,766 Timestamp Rollups a year
    - 24 per day
- Staking:
  - `~0.106` Capacity per Rollup
  - `~2.544` Capacity per Day
  - `50·2.544 = ~128` FRQCY Staked (50:1 currently)

OpenTimestamps: Cost for Three Calendars

- Average time between transactions in the last week:
  - 3 minutes
  - 175,320 Timestamp Rollups a year
    - 480 per day
- Staking:
  - ~0.106 Capacity per Rollup
  - ~51 Capacity per Day
  - 50·51 = ~2550 FRQCY Staked (50:1 currently)

OpenTimestamps: Questions

- *Discovery*: How do I want this data accessed?
- *Storage*: Where is the data stored?
- *Schema*: How is this data structured?

OpenTimestamps: Questions

- *Discovery*
  - Time-based or User-based?

OpenTimestamps: Questions

- *Discovery*: Time-based
- *Storage*
  - Has aggregation already
  - Single “Item” per time slot per calendar
  - Example (32 bytes):

    `2043d2463ed68083eae3d101aa3aa903435bd2c002d1d23df644b25bd7a4bda338`
  - Suggestion: Avro, On-Chain

OpenTimestamps: Questions

- *Discovery*: Time-based
- *Storage*: Avro, On-Chain
- Schema:
  - Raw Schema Type? (Not currently available)
  - Avro

OpenTimestamps Avro Schema

```
{
  type: "record",
  name: "OpenTimestamps",
  fields: [
    {
      name: "commitment",
      type: "bytes", // Avro doesn't specify length
      doc: "opentimestamps.org commitment 32-byte Merkle root"
    }
  ]
}
```

OpenTimestamps: Questions

- *Discovery*: Time-based
- *Storage*: Avro, On-Chain
- Schema:
  - Data: `[{name: "commitment", type: "bytes"}]`
  - Permissions: None

OpenTimestamps: Make It!

- Testnet Faucet

  `faucet.testnet.frequency.xyz`

- Extrinsic

  `schemas.createSchemaV3`

using the selected account
ALICE (EXTENSION)

free balance 4,027.3759 xrqcy
5GrwvaEF5...

submit the following extrinsic
schemas createSchemaV3(model, modelType, payloadLocation, settings, schemaName) createSchemaV3

model: Bytes
{"type":"record","name":"OpenTimestamps","fields":[{"name":"commitment","type":"bytes","doc":"opentimestamps.org commitment 32-byte Merkle root"}]}
file upload

modelType: CommonPrimitivesSchemaModelType
AvroBinary

payloadLocation: CommonPrimitivesSchemaPayloadLocation
OnChain

settings: Vec<CommonPrimitivesSchemaSchemaSetting> (Vec<SchemaSetting>)

Add item Remove item

schemaName: Option<Bytes> (Option<SchemaNamePayload>)
include option

Bytes
ots.commitment
file upload

OpenTimestamps: Make It!

- Testnet Faucet

  `faucet.testnet.frequency.xyz`

- Extrinsic

  `schemas.createSchemaV3`

- Mine

  `Id: 16296`

OpenTimestamps: Use It!

- Encoding

  `wilwade.github.io/avro-json/`

- Extrinsic

  `messages.addOnchainMessage`

- Mine

  `Id: 16296`

  `Data:`

`0x422043d2463ed68083eae3d101aa3aa90`

`3435bd2c002d1d23df644b25bd7a4bda338`

OpenTimestamps: Use It!

- Extracting

  `wilwade.github.io/avro-json/`

- RPC

  `messages.getBySchemaId`

- Mine

  `Id: 16296`

  `Block Range: 5333500-5333549`

  `Page Size: 1`

Interjection: OpenTimestamps “Native”?

OpenTimestamps: Frequency Native?

- Use IPFS to keep the list of all timestamps instead of using a Merkle root
- Offer direct-to-user timestamping on chain
- Other ideas?


Build Your Own!

Build Your Own References

- https://wilwade.github.io/avro-json/
- Extrinsic: `schemas.createSchemaV3`
- RPC: messages.getBySchemaId
- Frequency Schema Docs: https://frequency-chain.github.io/frequency/pallet_schemas
-

Thank you

www.frequency.xyz

---

docs.frequency.xyz

---

github.com/frequency-chain