

# Sketches of IPLD example apps & their data structures

Compiled 2018-06-27

## 0 Introduction

This document is compiled from a selection of five examples that were brainstormed during a session in Lisbon on 2018-05-13, linked from the bottom of [this Google doc](#). Each sketch fits on its own page. I (@davidad) have edited them somewhat, mostly to harmonize variations in the choice of syntax<sup>1</sup>.

Potential uses of this document include:

- as a ballpark sketch of the motivation and scope of IPLD’s medium-term ambitions
- as an inspiration for generating new open questions (“wait, how would *that* work?”)
- to provide a grounding, concrete reference point for consideration of future IPLD language features

Note that the final example, [IPFS Directories](#), involves traits (also known as interfaces or typeclasses), which may be a post-L0 feature.

---

<sup>1</sup>This is to facilitate comparisons across the different examples; the decisions of what kinds of syntax to use here have not been thought through carefully and are not intended as proposals about the default surface-syntax of L0.

# 1 Filecoin (orig. @dignifiedquire)

```
1 type Block = struct {
2     // reference to the previous block
3     // only `None`, for the genesis block
4     parent: Option<Cid<Block>>,
5     // hash of the statetree after execution of all included messages
6     state: Cid<StateTree>,
7     // a list of all state transitions this block entails
8     timestamp: DateTime,
9     messages: Cid<Vec<Message>>,
10    // a list of all receipts, for the messages included in this block
11    message_receipts: Cid<Vec<MessageReceipt>>,
12 };
13
14 impl Block {
15     fn Hash() -> Hash {}
16 };
17
18 type Message = struct {
19     // None if this message is not signed
20     signature: Option<Hash>,
21     // what is the Method type, would be great to tie in the exports with IPLD, not clear how
22     method: Method,
23     args: Vec<u8>,
24     to: Address,
25     from: Address,
26 };
27
28 type MessageReceipt = struct {
29     exit_code: u8,
30     result_ptr: u32,
31     result_size: u32,
32 };
33
34 type Address = Vec<u8>;
35 type Hash = Vec<u8>;
36
37 type StateTree = Map<Address, Actor>;
38
39 type Actor = struct {
40     balance: u64,
41     // how is storage modeled?
42     // could be just Vec<u8>
43     storage: Cid<Vec<u8>>,
44     public_key: Option<Vec<u8>>,
45 };
46
47 // How to model concrete actors vs the type actor that represents an actor on the host level?
48 impl Actor {
49     // is this right? how to do this?
50     fn Exports(&self) -> Vec<Method>;
51
52     // public exports go here, but they are not part of the actor
53     // need to mention actors' types, i.e. IPLD about IPLD types. not sure how to express this
54 };
```

## 2 Decentralized Twitter

```
1 type User = struct {
2   id: PubKey,
3   handle: Utf8String,
4   displayName: Utf8String,
5   bio: BoundedArray<Utf8Char, 140>,
6   profilePic: Hash<Image>,
7   coverPic: Hash<Image>,
8   tweet_HEAD: LatestSigned<TweetEntry, id>,
9   like_HEAD: LatestSigned<LikeEntry, id>,
10 };
11
12 type TweetEntry = signed struct {
13   parents: List<Hash<TweetEntry>>, // previous commits from this user
14   timestamp: DateTime,
15   tweet: enum Tweet {
16     PlainTweet: struct {
17       text: BoundedArray<Utf8Char, 280>,
18     },
19     ReplyTweet: struct {
20       inReplyTo: List<Hash<TweetEntry>>,
21       text: BoundedArray<Utf8Char, 280>,
22     },
23     PlainRetweet: struct {
24       retweetOf: Hash<TweetEntry>,
25     },
26     RetweetWithComment: struct {
27       retweetOf: Hash<TweetEntry>,
28       text: BoundedArray<Utf8Char, 280>,
29     },
30   },
31 };
32
33 type LikeEntry = signed struct {
34   parents: List<Hash<LikeEntry>>, // previous commits from this user
35   likeOf: Hash<TweetEntry>,
36 };
```

### 3 IPRS (orig. @diasdavid)

```
1 type SignedRecord = struct {
2     record: Record,           // record or link to record
3     Signature: Vec<u8>,       // signature of the record (Value + Scheme + ValidityData)
4     PublicKey: PublicKey,
5 };
6
7 type Record = struct {
8     Value: Vec<u8>,
9     Scheme: Cid,              // scheme to validate the record (expiry date, geographic location, etc)
10    ValidityData: ValidityData,
11 };
12
13 type ValidityData = struct {
14     ExpiryDate: DateTime,
15     GeoLocation: XYCoordinates, // XYCoordinates and range
16     Ancestry: Cid,              // only valid in the presence of another record
17     SeqNumber: u32,
18 };
19
20 type Identity = struct {
21     SecretKey: SecretKey,
22     PublicKey: PublicKey,
23 };
24
25 type SecretKey = Vec<u8>;      // possibly more than an array in the future (i.e Linked-Data Key / MultiKey)
26 type PublicKey = Vec<u8>;
```

## 4 Collaborative Real-Time Applications (orig. @diasdavid)

```
1 type User = struct {
2   KeyPair: KeyPair,
3   Details: Person, // http://schema.org/Person
4 };
5
6 type Capability = struct { // granting access to a private resource through symmetric crypto
7   CypherData: CID,
8   AccessKey: SharedKey,
9 };
10
11 type Authorization = struct // granting other users permission or self proving permission
12   IssuerSignature: Vec<u8>,
13   Receiver: PublicKey,
14 };
15
16 type KeyPair = struct {
17   SecretKey: SecretKey,
18   PublicKey: PublicKey,
19 };
20
21 type SecretKey = Vec<u8>; // possibly more than an array in the future (i.e Linked-Data Key / MultiKey)
22 type PublicKey = Vec<u8>;
23 type SharedKey = Vec<u8>; // symmetricKey
```

## 5 IPFS Directories (orig. @stebalien)

```
1 trait Iterable {
2     type Item;
3     fn map<T>(self, cb: fn(item: &Self::Item) -> T) -> Iterable<Item = T>;
4     fn skip(self, n: NonNegative) -> Iterable<Item = Self::Item>;
5     fn take(self, n: NonNegative) -> Iterable<Item = Self::Item>;
6 };
7
8 trait Collection: Iterable {
9     fn length(&self) -> NonNegative;
10 };
11
12 struct Range {
13     start: NonNegative,
14     length: NonNegative,
15 };
16
17 trait IndexedCollection: Collection + Map<Key=NonNegative, Value=Self::Item> {
18     fn slice(self, range: Range) -> Option<IndexedCollection>;
19     fn splice(self, range: Range, Collection<Item=Self::Item>) -> Option<Collection<Item=Self::Item>>
20 };
21
22 trait Map {
23     type Key;
24     type Value;
25
26     fn get(&self, name: Self::Key) -> Option<Self::Value>;
27     fn update(self, name: Self::Key, value: Self::Value) -> Option<Map<Key=Self::Key, Value=Self::Value>>;
28 };
29
30 trait FsObject {
31     fn created_at(&self) -> Time;
32     fn modified_at(&self) -> Time;
33 };
34
35 // Directories are collections of file objects.
36 //
37 // To *use* the file objects, you'll have to *cast* to a Directory, File, etc.
38 trait Directory:
39     Collection<Item = Cid<FsObject>> + Map<Key = String, Value = Cid<FsObject>> + FsObject
40 {
41 };
42 trait RegularFile: FsObject + Collection<Item = u8> {};
```