Publication forms

What a human might want to publish, where it comes from, and how they say so. This is the input side of `publishing-on-ethereum.md` — the end users, before any question of how a publisher tool is built. Nothing here is implemented; it is a frame for deciding what to implement.

Four kinds, and only four

Everything anyone publishes digitally reduces to **text, audio, static image, moving image**. The last two are both "visual" and it is tempting to collapse them, but they behave differently enough in every other dimension that keeping them apart earns its keep.

A **publication** is a *set of tracks*. Each track has a kind, an extent and a source. That is the whole model, and the rest of this document is the three dimensions and what falls out of them.

This matches the output side already described in `publishing-on-ethereum.md` — *"Input can be any number of texts, audio tracks and video tracks, in any combination. Output can also be any number of texts, audio tracks and video tracks."* What was missing was **where the input comes from** and **whether it ends**.

Dimension 1: extent — and it decides the topology

extent	means	publication topology
instant	no duration. a text, an image	a single manifest
finite	a known duration, known before publishing starts	a single manifest
unbounded	no end known at the start	**a feed, published incrementally**

This is the structural consequence, not a UX detail. If the publisher knows the extent before it begins, the publication can be one addressable object. If it does not, the publication must be a feed that grows — which is exactly the rolling playlist with a `prev` pointer per update that the null-audio path already produces.

The boundary is subtler than "live or not". "Record thirty seconds" is finite. "Record until I press stop" is *unbounded with intent to finish* — operationally a feed, which may be sealed into a manifest afterwards. The question is never "is it live", it is **"do you know the extent yet?"**

Dimension 2: source — and the device binding it implies

source	means	needs
stored	already a file on the host	a path
captured	produced now, by a device on the host, to a known extent	a device, and the extent
streamed	produced continuously by a device, no end	a device

Only `stored` needs nothing but a path. The other two bind to **a specific device**, and a host has several:

- **microphone** — and there may be more than one

- **camera** — front, back, external; a smartphone publisher has at least two
- **screen** — a screenshot is a captured static image, a screencast is a streamed moving image, from the same device
- **keyboard** — the overlooked one. Live text, delimited by return, is a stream of instants

That last row is why this frame covers conversation and not just media. A publisher typing into an unbounded text feed is chat. The same structure that publishes a radio show publishes a message thread.

Dimension 3: cardinality

Any number of any kind, in any combination. One text. Seventeen videos. Thirty-two audio files, an image from the camera and three texts, together, as one publication.

This is not a stress case to be tolerated; it is the normal case seen clearly. "A video with a title and a thumbnail" is already three tracks of three kinds, and nobody thinks of it as exotic.

Worked examples

Each row is a real thing a person wants to publish. The point of the table is that the left column is recognisable to a human and the right column is mechanical.

what a person says	tracks
a tweet	1 text · instant · keyboard
a photo with a caption	1 image · instant · camera(back) + 1 text · instant · keyboard
a screenshot with a note	1 image · instant · screen + 1 text · instant · keyboard
a voice memo	1 audio · finite · microphone
an EPUB to turn into an audiobook	1 text · instant · stored *(output: audio)*
a movie	1 video · finite · stored
a video upload with title and thumbnail	1 video · finite · stored + 1 text · instant · keyboard + 1 image · instant · stored
live radio	1 audio · **unbounded** · microphone
a live stream with a running commentary	1 video + 1 audio · unbounded · camera+mic + 1 text · unbounded · keyboard
a recorded lecture	1 video · finite · camera + 1 audio · finite · microphone + 1 video · finite · **screen**
a photo essay	N images · instant · mixed stored and camera + N texts
the stress case	17 video · stored + 32 audio · stored + 1 image · camera + 3 text · keyboard

The lecture row is worth noting: **two moving-image tracks from two different devices**, camera and screen, simultaneously. Any model that assumes one visual source per publication breaks there, and lectures are not rare.

What this buys: capability matching stops being a guess

Once a publication declares its tracks, finding somewhere to process it is mechanical rather than editorial:

```
1. read the ServiceRegistry onchain          -> orchestrator addresses and service URIs
2. GET /discovery on each, out of band       -> the apps each one runs, capacity, price
3. keep those whose capability signature     -> a capability declares what it ACCEPTS
   accepts this publication's track kinds       and what it EMITS, in the same four kinds
4. choose on price, capacity, proximity
```

So a capability is typed by its input and output signature. "Accepts one audio, emits one text" is captions. "Accepts one text, emits one audio" is an audiobook. "Accepts one video and one audio, emits N video and N text" is transcoding with captions. The publication declares a shape; discovery finds the shapes that fit.

That is the part worth building toward, because it is the step that makes the network usable by someone who does not know what an orchestrator is.

Open questions

Is "role" a separate axis from "kind"? In the video-with-title-and-thumbnail case, the video is the subject and the title and thumbnail *describe* it. Both are tracks, but they are not peers. Either roles are a label on a track (`subject`, `title`, `poster`, `caption`) or the model stays flat and the capability's signature carries the meaning. Flat is simpler and probably right first; the risk is that every capability reinvents "which text is the title".

How do tracks relate in time? For finite and unbounded multi-track publications the tracks must share a timeline, or captions land on the wrong frames. The captions work already solves this by sharing the input stream's PTS. Instant tracks have no timeline, so a publication mixing instant and finite tracks has a timeline for some of its tracks and not others.

Who decides the extent of an unbounded capture? The publisher pressing stop, a duration set at the start, or the capability's price running out. All three happen, and they are not the same event.

Does a sealed feed become a manifest? An unbounded publication that has finished is indistinguishable, afterwards, from a finite one — except that it was published as a feed. Sealing it into a single manifest at the end would make it addressable as one object, at the cost of publishing the content's index twice.

Where does selection live? "Which camera" is a device question, but "which 30 seconds of this video" is an editing question. The frame above has no place for trimming, and real publishers trim.