PACT – Permanent Anchored Computation Theory

A Minimal Protocol for Identity and Anchored Action on Bitcoin via Ordinals

OrdinalsLover
`@blockapunk`

April 28, 2026

PACT is an ontology for identity and anchored action. Its general pattern is four-part: namespace, world, avatar, and transition. PACT v1 specifies one concrete instantiation of that pattern on Bitcoin Ordinals. Bitcoin provides permanence and ordering; Ordinals makes that permanence individually addressable, browser-reachable, and practically usable without privileged services. This opening makes sat-bound identity possible: a specific satoshi can anchor a persistent entity whose inscriptions and transitions remain independently verifiable across time. This document specifies the PACT v1 envelope, its object rules, Merkle-based verification methods, and the relation between active PACT v1 objects and historical predecessor schemas.

Contents

1 Introduction **4**
- 1.1 What PACT is . . . 4
- 1.2 What PACT is not . . . 4
- 1.3 Notation and conventions . . . 4

2 Substrate **6**
- 2.1 Bitcoin as persistent anchor . . . 6
- 2.2 Sat numbering and ownership . . . 6
- 2.3 Sat as identity . . . 6
- 2.4 Ordinals as transport layer . . . 8

3 The four-part ontology **10**
- 3.1 The pattern in general . . . 10
- 3.2 Namespace as organising container . . . 10
- 3.3 World as defined unit of rules . . . 11
- 3.4 Entity as instantiation within worlds . . . 11
- 3.5 Action as recorded transition . . . 11
- 3.6 How the parts compose . . . 12
- 3.7 Where this pattern appears . . . 12
- 3.8 PACT as one instantiation . . . 12

4 Object Types **14**
- 4.1 Namespace variety . . . 14
- 4.2 Optional world . . . 15
- 4.3 Avatar . . . 15
- 4.4 Transition . . . 16

5 Envelope Form **18**
- 5.1 PACT v1 JSON structure . . . 18
- 5.2 Required fields . . . 18
- 5.3 Optional fields . . . 18
- 5.4 Examples . . . 19

6 Merkle Tree Construction (RFC 6962) **20**
- 6.1 Motivation . . . 20
- 6.2 Definition . . . 20
- 6.3 Leaf hashing with domain separator 0x00 . . . 20
- 6.4 Internal node hashing with domain separator 0x01 . . . 21
- 6.5 Merkle root computation . . . 21
- 6.6 Audit path verification . . . 21

7 State Transition **22**
- 7.1 Motivation . . . 22
- 7.2 Formal definition . . . 22
- 7.3 Operational description . . . 22
- 7.4 Validity conditions . . . 23

8 Verification **24**
- 8.1 Browser-side verification . . . 24
- 8.2 Independent parser verification . . . 24
- 8.3 Audit path verification walkthrough . . . 25

9 Security Considerations **27**
9.1 What PACT guarantees . . . 27
9.2 What PACT does not guarantee . . . 27
9.3 Attack scenarios and mitigations . . . 29

10 A working demonstration: OrdinalsWall **31**
10.1 What OrdinalsWall is . . . 31
10.2 The structure of the site . . . 31
10.3 Verifiable references throughout the site . . . 32
10.4 How the site embodies the specification . . . 33
10.5 What the site does not claim to be . . . 33
10.6 Avatar perspective: cycles as cognition in time . . . 34
10.7 Historical predecessors as visible heritage . . . 34

11 Intellectual context **35**
11.1 Ontology versus database . . . 35
11.2 Why this matters for Bitcoin Ordinals . . . 35
11.3 Why ontology-based systems are difficult to build . . . 36
11.4 The traditions PACT draws on . . . 36
11.5 Beyond Ordinals: PACT as identity anchor . . . 37
11.6 What this combination produces . . . 37

12 Adoption, usability, and portability **39**
12.1 Adoption: who can build under PACT . . . 39
12.2 Usability: a familiar object-oriented form . . . 39
12.3 Portability: a Model-View-Controller separation . . . 40
12.4 What this combination produces . . . 41
12.5 Computation as a future protocol layer . . . 41
12.6 Financial transactions as a parallel example . . . 42

13 Acknowledgements **43**

A Pseudocode **44**
A.1 Envelope validation . . . 44
A.2 Merkle root computation (RFC 6962) . . . 44
A.3 Transition validity check . . . 44

B Worked Example **46**
B.1 Wanderer-001 Cycle 1 envelope . . . 46
B.2 Verification walkthrough step by step . . . 46

1 Introduction

1.1 What PACT is

PACT, short for *Permanent Anchored Computation Theory*, is an ontology for identity and anchored action. The ontology itself is general: the same four-part pattern of namespace, world, avatar, and transition appears wherever systems must account for who acts within what context, with what consequences, in what order. PACT v1 is one concrete realization of that pattern, specified here for Bitcoin Ordinals.

What gives this realization its particular freedom is the substrate it uses. Bitcoin provides permanence, neutral ordering, and a history no operator can rewrite, but those properties are not by themselves directly reachable for most readers. Ordinals — Casey Rodarmor's [3] playful protocol for attaching inscriptions to individual satoshis — makes that permanence individually addressable and browser-reachable. Through Ordinals, a PACT object can be followed by URL, inspected in ordinary tools, and verified without asking any service for permission.

The recognition that individual satoshis can serve as identity anchors follows from this opening. When sats are individually addressable and chain-permanent, they can carry identity that no service can revoke and no naming system can let lapse. Bitoshi's [6] Bitmap Signal Theory recognized an early form of this principle for spatial claims on Bitcoin; PACT extends it to identity in a fuller sense: sat-bound, context-bearing, and persistent across time.

PACT is therefore not simply "a protocol on Bitcoin." More precisely, it is an ontology that becomes practically free through Ordinals, with Bitcoin as the permanence layer on which Ordinals lives. Across the source material, one formulation appears repeatedly: sat equals identity, inscription equals state, transaction equals transition, and flow equals proof. The protocol is therefore not a smart-contract runtime in the Ethereum sense. It is a disciplined interpretation layer over existing chain data.

In the older Bitronaut notes, this appears as the claim that "state is not stored; state is derived." In the checkpoint specification, the same principle appears as byte-exact canonicalization, deterministic hashing, and independent re-computability of Merkle roots. In the parser code, it appears as a decoding and validation pipeline that aims to return the same result for any implementation that applies the same rules to the same Ordinals and Bitcoin inputs.

1.2 What PACT is not

PACT is not itself an application, game, social network, or website. It is not tied to a single world, a single parser implementation, or a single user interface. It is also not identical to legacy machine-centric JSON payloads found in early Bitronaut files. Those earlier payloads are historically important, because they show the transition from broad conceptual framing to a cleaner envelope model, but they are not normative for the present document.

PACT is also not a claim that Ordinals defines semantic truth. Ordinals only supplies an inscription transport and a sat-based identity surface. Interpretation, validity conditions, and state evolution are PACT concerns.

1.3 Notation and conventions

This specification uses the following conventions:

- `PACT` written in uppercase denotes the required literal protocol identifier in the active PACT v1 envelope.
- A *sat* is a satoshi interpreted under ordinal numbering.
- A *carrier sat* is the satoshi whose history defines the persistent identity of a PACT object.

- JSON field names are shown in monospace, for example `avatar` or `merkle_root`.
- Raw bytes are distinguished from hex strings. Concatenation in hash formulas is byte concatenation unless explicitly stated otherwise.
- The words *MUST*, *SHOULD*, and *MAY* are used in their ordinary specification sense.

2 Substrate

2.1 Bitcoin as persistent anchor

Bitcoin contributes three properties that matter to PACT:

1. global ordering by block height and transaction order,
2. durable witness-carrying transactions,
3. a UTXO model in which satoshis move through transactions under FIFO assignment.

PACT does not ask Bitcoin to execute arbitrary code. Instead, it uses Bitcoin as a persistence layer and ordering oracle. On its own, however, Bitcoin does not expose those properties in a form most readers can inspect object by object. Ordinals opens that substrate by making individual sats and inscriptions addressable in ordinary tools. If two parties inspect the same blocks and apply the same protocol rules, they should derive the same protocol state.

2.2 Sat numbering and ownership

The identity primitive in PACT is not an account address or a mutable database row. It is a satoshi followed through the Bitcoin UTXO graph. Let a transaction input set be ordered as Bitcoin defines it, and let each input contribute a value in satoshis. The absolute sat index of a sat within a transaction input stream is:

$$\text{absoluteSatIndex} = \sum_{i<k} \text{value}(I_i) + \text{offset},$$

where I_k is the input carrying the sat and offset is its zero-based position inside that input's sat range.

This rule appears in the source material as the operational summary "inputs to FIFO to outputs." In PACT, the same sat reappearing in a later inscription context is the basis for same-sat identity. The sat is the durable identity; individual inscription bodies are merely successive states or declarations associated with that identity. Bitcoin supplies the movement rule; Ordinals makes the moving sat individually addressable, inspectable, and linkable for ordinary readers.

2.3 Sat as identity

The central claim of PACT is that a satoshi is a sufficient substrate for persistent identity. This subsection makes the claim explicit, explains why it holds, describes how it is enforced, and notes what follows from it.

Definition 1 (Sat-bound identity)**.** *A* sat-bound identity *is the unique persistent reference formed by a single satoshi, identified by its absolute ordinal number, together with the sequence of inscriptions and transactions that have moved it through the Bitcoin UTXO graph from its first appearance to the present chain tip.*

Why a satoshi is suitable. Three properties of satoshis under ordinal numbering combine to make them suitable as identity anchors:

1. *Permanence of numbering.* Every satoshi has a unique ordinal number determined by its mining order, fixed since the Bitcoin genesis block. This number cannot be revoked, reissued, or remapped without invalidating the Bitcoin chain itself.

2. *Verifiable ownership through the UTXO graph.* The current controller of a satoshi is determined by which unspent transaction output contains it. Ownership transfers happen through ordinary Bitcoin transactions and are recorded in the same chain that defines the satoshi's history. No external registry is consulted.

3. *Independence from any authority.* No party, including the original inscriber, controls a satoshi after it has moved. There is no issuer, no custodian, no registrar who can intervene in the identity it carries.

These three properties together make the satoshi a primitive that other identity systems lack. PGP keys can expire and require revocation infrastructure to be honored. DNS names are leased through registrars who can refuse renewal. Nostr public keys provide cryptographic authorship but no anchoring to a durable substrate; if all relays storing a public key's events disappear, the key remains valid but has nothing to verify against. A sat-bound identity inhabits none of these failure modes.

Bitcoin alone supplies the permanence behind this claim, but Ordinals is what makes the claim practically available. By binding inscriptions to individually addressable sats and making those inscriptions reachable through ordinary browser tools, Ordinals turns a latent property of Bitcoin into a usable identity surface. PACT depends on that opening. Without it, sat-bound identity would remain conceptually interesting but operationally remote.

How sat-bound identity is enforced. The protocol does not enforce identity in the sense of granting or denying access. It enforces identity by making continuity verifiable. Three mechanisms together provide this:

1. *Inscription on a specific satoshi.* An avatar's constituting inscription is bound to a specific satoshi chosen by the inscriber. After the reveal transaction confirms, that inscription is permanently associated with that satoshi in any conforming Ordinals interpretation.

2. *Same-sat continuity verification.* Subsequent transitions claiming to act on behalf of an avatar must be verifiable as carried by, or referring to, the same satoshi. A parser determines this by following the satoshi through the UTXO graph from the constituting inscription forward, applying the same ordinal-numbering rules used by Bitcoin.

3. *No external authority.* The protocol does not consult any indexer, explorer, or registrar to validate identity. Two independent parsers operating on the same Bitcoin chain data must arrive at the same conclusion about which transitions belong to which avatar.

The result is that identity is enforced by reproducibility, not by permission. A claim of identity that does not survive independent re-verification against the chain is not a valid claim, regardless of how it is presented.

Consequences. Several properties follow from sat-bound identity that do not hold for other identity systems:

- *Identity is transferable but not revocable.* Because identity is bound to satoshi ownership, transferring the satoshi transfers the identity. There is no separate revocation channel; the only way to relinquish an identity is to transfer the satoshi to another party or to a provably unspendable output.

- *Identity is anonymous by default.* A satoshi discloses no information about who controls it. Inscriptions on the satoshi may include human-readable names or affiliations, but the protocol does not require them. An avatar can act under a sat-bound identity without revealing the legal or social identity of its operator.

- *Identity persists beyond infrastructure.* Sat-bound identity does not depend on any service, server, or organization remaining operational. As long as Bitcoin persists, the identity remains verifiable, even if the inscriber is no longer reachable or no longer alive.
- *Identity is not equivalent to legal personhood.* A sat-bound identity refers to a unique entity with a verifiable history, but it does not establish legal standing, accountability, or jurisdiction. These remain matters for systems outside PACT.

The remainder of this section completes the substrate picture with Ordinals as transport layer; the rest of the specification builds on sat-bound identity without further argument.

Figure 1 contrasts the inscription strategy of Ordinals (any sat will do) with that of PACT (the identity sat is always carried at `input[0]`).

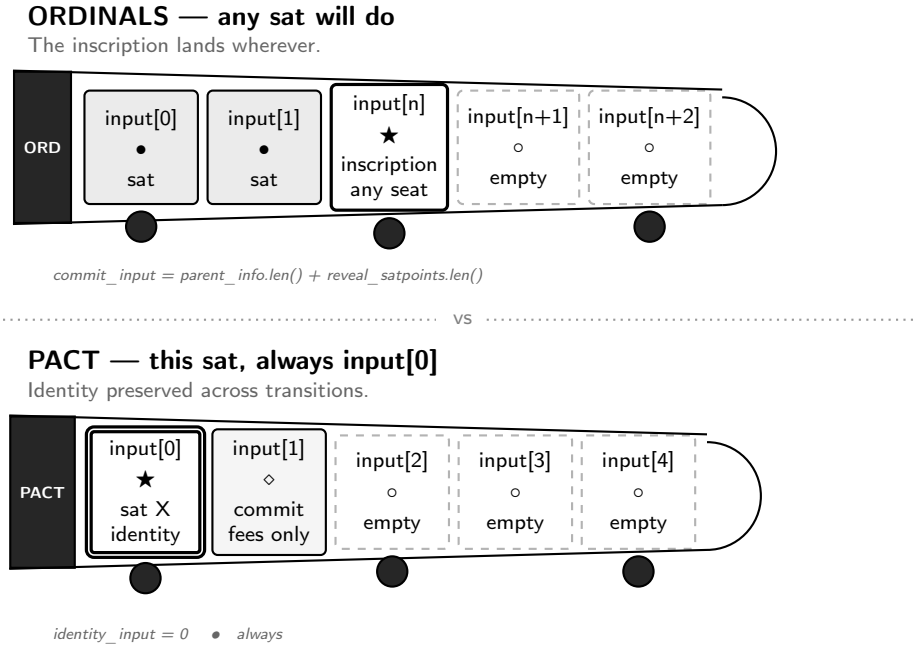

Figure 1: Two inscription strategies. In Ordinals (top), the inscription may be carried by any input; the choice is determined by ergonomics, not protocol meaning. In PACT (bottom), the identity sat is always at `input[0]`, ensuring that ordinal position rules deliver it to `output[0]` unchanged. The same satoshi enters and leaves the transaction; only its inscription state evolves.

2.4 Ordinals as transport layer

Ordinals is the transport mechanism that places arbitrary bytes into witness data and associates inscriptions with sats [3]. But for PACT its role is more than carriage alone. Ordinals is the layer that makes Bitcoin's permanence practically free: it makes individual sats addressable, inscriptions browser-reachable, and chain history inspectable without privileged software. The parser materials are explicit that Ordinals does not define PACT validity by itself. Semantics, envelope rules, and state evolution remain PACT concerns. Yet without Ordinals, those concerns would not be exposed to ordinary readers in the same way.

This is why the distinction matters. Bitcoin is the durable tool underneath: it orders transactions, preserves witness bytes, and enforces UTXO continuity. Ordinals is the opening through which that durability becomes individually readable in the world. PACT relies on both, but in different roles. Bitcoin provides the permanence; Ordinals makes the permanence addressable and practically accessible.

The practical result is a layered picture:

Layer	Role
Bitcoin	ordering, persistence, UTXO flow
Ordinals	envelope transport, sat association
PACT	object model, validity rules, verification

3 The four-part ontology

PACT organises information through four object types: namespace, world, avatar, and transition. Before defining these in their PACT-specific form, this chapter describes the underlying pattern. The pattern is not unique to PACT or to Bitcoin. It appears wherever a system needs to account for identity, context, and recorded action – software architectures, knowledge representation, legal systems, biological taxonomy, online platforms. PACT recognises this pattern and applies it through Bitcoin Ordinals, but the pattern itself is older and broader than any single implementation.

3.1 The pattern in general

A system that records what entities do within defined contexts must answer four questions:

- How are related definitions grouped together?
- Within a group, what rules apply?
- Who acts under those rules?
- What did they do?

The four-part ontology answers these in turn. A *namespace* is the grouping. A *world* is the definition of rules within a namespace. An *avatar* (or, more generically, an entity) is who acts within a world. A *transition* (or, more generically, an action) is what was done.

This is not a novel decomposition. It is what software engineers, ontologists, and system designers have converged on whenever they need to describe identity and action at scale. What is unusual about PACT is not the pattern but the substrate it places the pattern on.

3.2 Namespace as organising container

A namespace is a name-bearing container for related definitions. It has no executable behaviour of its own. Its job is to mark what belongs together and to distinguish that grouping from other groupings.

The pattern appears throughout computing and beyond:

- Java packages: `java.util`, `com.example.payments`.
- DNS domains: `example.com`, `wikipedia.org`.
- RDF vocabularies: `foaf:`, `schema:`, `dc:`.
- File system folders: `/home/user/projects`, `/var/log`.
- Biological taxonomic ranks: kingdom, phylum, class.

What unites these examples is that the namespace itself does nothing. It declares boundaries; it invites contents; it is the address under which other definitions can be reached. A namespace named `commerce` does not transact; it indicates that what is found within it concerns commerce.

3.3 World as defined unit of rules

A world is a defined set of rules and vocabularies that applies within a namespace. Where a namespace organises, a world specifies. It says: within this context, these actions are valid; these data shapes are expected; these constraints must hold.

A world is not a template that produces instances. It is a single, uniquely defined unit. Two worlds with the same name in the same namespace would be a conflict, not two instances of one type.

The pattern appears across domains:

- Programming language modules: a single file or unit defining functions and types together.
- Game rule books: chess defines its rules once; every chess game follows the same definition.
- Legal jurisdictions: Dutch law is a defined body of rules; one cannot have two competing Dutch laws in the same context.
- Mathematical structures: a particular group, a particular ring – defined by its axioms.
- Social platforms: each platform defines what posts, comments, and connections mean within it.

The world is the answer to "what counts as valid here?"

3.4 Entity as instantiation within worlds

An entity is a concrete particular that exists within one or more worlds. Entities are where instantiation happens: many entities can exist in the same world, each with its own identity, history, and capacity to act. Two entities sharing a world share its rules but remain otherwise independent.

The pattern is widespread:

- A class instance in object-oriented programming: many objects of the same class coexist, each with its own state.
- A user account on a platform: many users follow the same platform rules, but each has its own identity and history.
- A citizen under a legal jurisdiction: many citizens, one body of law.
- A node in a graph database: many nodes, one schema.
- An organism in an ecosystem: many organisms, one set of ecological constraints.

In PACT, this role is filled by the avatar. The term "avatar" is chosen to emphasise that these are identifiable, persistent presences with continuity across time, not transient records. But the underlying role is the role of any instantiated entity within a defined context.

3.5 Action as recorded transition

An action is what an entity does within a world, recorded in a way that remains visible afterwards. An action has four dimensions: the entity that performed it, the world in which it occurred, the kind of action performed, and the moment at which it took place.

Examples appear in every system that records what happens:

- A method invocation: an instance receives a message, executes, and the call is logged.

- A user action on a platform: a post is made, a comment is added, a friend is added.
- A legal transaction: a contract is signed, a sale is registered.
- A move in a game: a piece is moved, the move is recorded.
- A biological event: an organism reproduces, feeds, dies.

In PACT, this role is filled by the transition. Transitions are the medium through which avatars exist beyond their constitution: an avatar that never acts has presence but no history.

3.6 How the parts compose

The four parts compose into a system. A namespace contains worlds. Each world defines rules under which entities act. Entities exist within worlds; entities produce actions; actions are recorded in chronological order, building up a history that can be read back at any time.

The composition has a direction. Namespaces are static and structural. Worlds are static within their definitions but invite activity. Entities are persistent but are dynamic in their participation. Actions are events: they happen, they are recorded, they cannot be undone.

What makes this composition powerful is that any of the four parts can be inspected independently. One can ask: what namespaces exist? What worlds live in this namespace? What entities populate this world? What actions has this entity performed? Each question addresses one part of the ontology, but together they yield a complete picture of a system that records identity and action over time.

3.7 Where this pattern appears

The four-part ontology is not invented by PACT. It is recognised by PACT. Versions of it appear in every domain where systems must account for who is doing what, where, and when.

In computer science, the pattern shows up as package/module/instance/method-invocation. In knowledge representation, it appears as namespace/ontology/individual/property-assertion. In biology, it manifests as kingdom/species/organism/event. In law, it takes the form of jurisdiction/code/citizen/act. In online services, it becomes platform/community/user/post.

That the same shape recurs across such different contexts suggests it is not arbitrary. It corresponds to the way human and artificial systems alike must organise themselves to track identity and action over time. Any system that records who did what, where, and when ends up with something close to this four-part structure, whether or not its designers explicitly recognise it.

3.8 PACT as one instantiation

PACT v1 instantiates the four-part ontology using Ordinals on Bitcoin. The ontology itself does not require Bitcoin; it requires a substrate that supports persistent, individually addressable units of inscription. Ordinals provides this on Bitcoin today. A future implementation might use a different substrate, provided the substrate offers the same essential properties: permanence, addressability, and accessibility without privileged services.

What makes Ordinals particularly suitable is not only that it exists on Bitcoin, but that it makes Bitcoin's permanence reachable through ordinary tools. A reader with a web browser can inspect any PACT object, follow its identity sat, verify its Merkle constructions, and read its history without trusting any intermediary service. The freedom that PACT depends on – the freedom from revocation, from registry capture, from service expiration – is delivered to the reader by Ordinals, not by Bitcoin alone.

The ontology and the substrate are therefore distinct but complementary. The ontology defines the shape of what is recorded; the substrate makes what is recorded permanent and

free. Future implementations on other substrates would require their own substrate-specific specifications, but they would inherit the same ontology described in this chapter.

The next chapter defines the PACT v1 implementation in its specific form: what envelope shape, what required fields, what validity rules. Reading it with the general pattern in mind clarifies which choices are fundamental to the pattern and which are specific to this implementation.

4 Object Types

The previous chapter described the four-part ontology as a general pattern. This chapter defines how PACT v1 instantiates that pattern through Ordinals on Bitcoin. The four object types – `NAMESPACE`, `WORLD`, `AVATAR`, `TRANSITION` – correspond directly to the four parts described generally above, but they are now specific: each has required fields, an envelope shape, and validity rules that apply to PACT v1 specifically.

4.1 Namespace variety

A namespace in PACT is a named context that groups one or more worlds. The active envelope model treats `NAMESPACE` as a top-level object type and `ns` as a field carried by worlds, avatars, and transitions. Where a world declares a specific semantic environment, a namespace declares the larger family of related environments that share an identifying label.

A namespace name is chosen by the inscriber and need not include the word PACT or any other prefix. The only protocol requirement is that the chosen name be valid as a JSON string. There is no registry, no authority, and no committee. The first inscription to declare a namespace name claims that name; subsequent inscriptions attempting the same name will not displace the original. This is the same first-claim discipline that governs sat-bound identity, applied at the level of organizational labels.

The namespace is the most open-ended object type in PACT. It is where the protocol meets the imagination of its users. The following examples are illustrative and do not all currently exist on chain; they show the range of contexts a namespace might organize:

- `pact-semantics` — for worlds concerned with structured public knowledge, formal vocabularies, and Semantic Web sources. The first example, currently on chain, contains the world `open-field`.
- `open-archive` — for worlds that anchor cultural-historical provenance: museum digitizations, documentary collections, archival metadata. Each world might correspond to a single institution or a thematic collection.
- `art-collective` — for groups of artists who wish to anchor authorship and provenance of digital works. Worlds within such a namespace could correspond to specific collectives, exhibitions, or eras.
- `seven-gardens` — for a hypothetical community of gardeners who anchor seasonal observations of plant life, soil conditions, and biodiversity. Worlds might correspond to individual gardens or regional climates.
- `sovereign-utterance` — for worlds that anchor public statements made under cryptographic identity, including Nostr events and similar attestation channels.
- `autonomous-agents` — for worlds that govern actions taken by autonomous software agents, with vocabularies for capability declaration, task acceptance, and result delivery.

These examples are not exhaustive, prescriptive, or competitive with one another. PACT does not adjudicate which namespaces are worth declaring or which are frivolous. A namespace claimed in good faith for a small community of three people is as valid under the protocol as a namespace claimed for an industrial consortium. The protocol provides only the mechanism; the meaning emerges from those who use it.

What namespace declaration accomplishes, then, is twofold. First, it reserves a label on the chain so that subsequent worlds, avatars, and transitions can refer to a stable parent context. Second, it makes visible to any reader of the chain that a particular work-area exists. The reader

may not understand or agree with the chosen domain, but the existence of the namespace is recorded. Future participants in similar work can build under the same namespace if its custodian permits, or declare their own adjacent namespace if they prefer independence.

The variety of possible namespaces is, in this sense, the variety of possible communities of practice that wish to record their work on a substrate that no party owns. A namespace is not a trademark; it is an invitation. The protocol cannot guarantee that the invitation will be accepted by anyone, only that the declaration will persist for as long as Bitcoin does.

4.2 Optional world

A world is an optional grouping context. The active envelope model treats `WORLD` as a top-level object type and `world` as a field carried by avatars and transitions. This allows protocol participants to define domain-specific environments without making worlds mandatory for the most minimal conceptual explanation of PACT.

A world's primary function is to define the vocabulary of transitions that are valid within it. The previous subsection described the TRANSITION envelope as the open end of the protocol: any kind of change can be carried, provided the envelope is structurally valid. A world narrows that openness for its own purposes. It declares which `action` values are recognized inside the world, what `data` fields are expected for each action, and what validity rules apply to the contents.

For a Semantic Web world such as `open-field`, the vocabulary includes actions like `OBSERVE` and `DECLARE`, with `data` fields for source IRI, source hash, and verdict. For a sovereign-utterance world such as one over Nostr, the vocabulary might include `ATTEST` with `data` fields for event identifier, public key, and content hash. For a world of autonomous agent activity, the vocabulary might include `ACT` or `REGISTER` with `data` fields for action type, artifact hash, and counterparty reference. Each world defines what kind of change makes sense in its domain, while the envelope itself remains the same.

A world also provides a context for avatars. Two avatars in the same world share a vocabulary; they can recognize each other's actions, respond to each other's transitions, and operate as participants in a common semantic space. Two avatars in different worlds are not required to share anything beyond the PACT protocol itself.

The local sources show two distinct roles for worlds:

- historical Bitronaut world-like structures tied to machine state narratives,
- the more explicit `pact/open-field` style world model reflected in the parser scaffolds and Wanderer files.

In this document, a world is treated as an optional protocol object that groups related avatars and transitions under a named semantic context, and that defines the action vocabulary recognized within that context. Worlds are not required for PACT to function; an avatar can act under PACT without belonging to any declared world. Worlds become useful when multiple participants want to share a common interpretation of what their transitions mean.

4.3 Avatar

An avatar is a sat-bound identity capable of participating in worlds through transitions. In the active PACT parser model, avatars are first-class object types with names, namespace membership, an originating world, and a `data` object that includes a sat reference. Wanderer-001 is the clearest concrete example in the local source corpus: its CONSTITUTE record ties the avatar to sat `1168224522821204` and to the `open-field` world.

An avatar is therefore not merely a name string. It is a protocol object whose continuity depends on following a specific sat and validating the transitions attached to that sat.

The avatar is in one important respect different from the other PACT object types. A namespace, a world, and a transition each belong to a single context: a namespace is one organizational unit, a world is one semantic environment, a transition is one action at one moment. An avatar is bound to a sat, not to a world. The constituting inscription declares an originating world for context, but the same sat can carry transitions in any number of worlds concurrently, provided those worlds accept the avatar's participation. This property follows directly from sat-bound identity: identity is anchored to the satoshi, not to the semantic context in which the satoshi first appeared, and a satoshi exists in only one state at a time regardless of how many worlds reference it.

The practical consequence is that an avatar can be present in multiple worlds simultaneously. Wanderer-001 is constituted in `open-field`, observing Semantic Web sources. The same avatar, identified by the same sat, can issue an attestation in a sovereign-utterance world and perform an action in an autonomous-agent world during the same period. Each transition declares its own world; the avatar exists in all of them at once. What ties these activities together is not sequential migration but the continuity of the underlying sat through the UTXO graph: one satoshi, one identity, many worlds.

Because the avatar is the same in every world it enters, transitions in one world can reference transitions in another. An attestation in a sovereign-utterance world may cite an observation made earlier in `open-field`; an action in an autonomous-agent world may depend on an earlier checkpoint. These are self-references across worlds, made possible by the fact that the satoshi is shared. An avatar can therefore reason about and act upon its own activity regardless of which world that activity took place in. This becomes particularly relevant for autonomous agents whose work spans domains: an agent that observes Semantic Web sources, attests to claims in public utterance channels, and executes actions on behalf of an operator can do all three under one identity, with each world's transitions verifiable from the others.

Whether a given world accepts a visiting avatar is a matter for that world's rules, not for the protocol itself. A world may choose to recognize only avatars constituted within it, or to accept any avatar whose constitution can be verified. The protocol leaves this discretion to world authors. What it guarantees is the identity itself: the sat is the same, regardless of where it acts, and regardless of how many worlds it acts in.

4.4 Transition

A transition is a protocol action carried in a PACT envelope. In the active envelope model, transitions include `ns`, `world`, `avatar`, `action`, and a `data` object. The parser source lists known actions such as `GENESIS`, `OBSERVE`, `CHECKPOINT`, `DECLARE`, `REJECT`, `DELEGATE`, and `TERMINATE`. Unknown actions may be accepted as protocol extensions, but their envelope must still be structurally valid.

The transition is the open end of the protocol. Avatar and world are relatively fixed — an avatar is constituted once on a sat, a world is declared once with its rules. Transitions are where ongoing change lives. Any kind of change that an avatar wishes to anchor through Ordinals on Bitcoin can be carried inside a transition, provided the envelope remains structurally valid. Different worlds will require different vocabularies of action: an observation in one world is not the same kind of act as a measurement in another, or an attestation in a third. The protocol does not prescribe which actions matter; it prescribes only the envelope that allows them to be recognized, ordered, and verified.

This is intentional. PACT v1 is meant to be a small ontology and envelope over which many different worlds can operate. A world for Semantic Web observation needs vocabulary for sources, hashes, and inferences. A world for sovereign utterances needs vocabulary for events, signatures, and attestations. A world for autonomous agent activity needs vocabulary for actions, mandates, and counterparty references. All three can be carried by the same TRANSITION envelope; only the contents of `action` and `data` differ. The protocol remains stable while the worlds it supports

diverge.

Wanderer's transition grammar adds a domain-specific vocabulary for off-chain and on-chain acts, but the protocol-level concept remains simple: a transition is a state-relevant action, attached to a concrete identity context, which can be validated and interpreted without trusting a privileged server.

5 Envelope Form

5.1 PACT v1 JSON structure

The active parser code describes the reference PACT v1 envelope as:

Listing 1: Reference PACT v1 envelope shape inferred from the active parser and Wanderer builder

```
{
  "p": "PACT",
  "v": 1,
  "type": "NAMESPACE | WORLD | AVATAR | TRANSITION",
  "ns": "<namespace name>",
  "world": "<world name>",
  "avatar": "<avatar name>",
  "action": "<transition action>",
  "name": "<entity name>",
  "data": { }
}
```

This structure is the normative envelope model for PACT v1 in this specification. The ontology described in Section 3 is broader than any one envelope format; what is fixed here is the concrete v1 realization. It supersedes lowercase `"p":"pact"` payloads and older machine-centric payloads where the central identity field was `machine` rather than `avatar`.

5.2 Required fields

The following fields are common and normative:

- `p`: MUST be the exact uppercase string `"PACT"`.
- `v`: MUST be the integer `1`.
- `type`: MUST be one of `NAMESPACE`, `WORLD`, `AVATAR`, or `TRANSITION`.

Type-specific requirements follow the parser's active validation logic:

- `NAMESPACE`: requires `name` and object-valued `data`.
- `WORLD`: requires `name`, `ns`, and object-valued `data`.
- `AVATAR`: requires `name`, `ns`, `world`, and object-valued `data`; current parser logic also requires `data.sat`.
- `TRANSITION`: requires `ns`, `world`, `avatar`, `action`, and object-valued `data`.

5.3 Optional fields

The envelope model is intentionally small. Optional semantics are carried inside `data`. For example, the Wanderer scaffolds use:

Listing 2: Observed extension fields in the Wanderer scaffold

```
{
  "semantic_ground": "sat:1168224522846323",
  "protocol_genesis": "sat:1168224522986323",
  "source_iri": "https://www.wikidata.org/wiki/Q30249683",
  "source_hash": "sha256:..."
}
```

This arrangement keeps the protocol envelope stable while allowing worlds and avatars to specialize the meaning of `data`.

5.4 Examples

Historical predecessor example. The local file `pact-genesis.json` preserves an early uppercase PACT example:

Listing 3: Historical predecessor from pact-genesis.json

```
{
  "p": "PACT",
  "v": 1,
  "type": "GENESIS",
  "machine": "machine-001",
  "created": "2026-04-06",
  "author": "blockapunk",
  "address_space": "linear"
}
```

This example is useful historically because it shows uppercase `PACT` and a clear protocol ambition, but it is not normative for the present envelope because `GENESIS` is outside the currently validated type set and the object is machine-centric rather than avatar-centric.

Companion checkpoint artifact example. The file `cycles/1/checkpoint.json` is not itself a PACT envelope. It is an off-chain checkpoint artifact that demonstrates how a cycle is summarized for anchoring:

Listing 4: Cycle 1 checkpoint artifact

```
{
  "schema": "wanderer-checkpoint/v1",
  "cycle": 1,
  "avatar": "wanderer-001",
  "merkle_root": "sha256:
    d0082b508e01d45dee0dbccf6e8950dc039b4030c546aa475836b0008feccc71",
  "leaf_count": 4,
  "tree_uri": "https://wanderer.bitronaut.io/cycles/1/tree.json",
  "worklog_uri": "https://wanderer.bitronaut.io/cycles/1/utterances.jsonl"
}
```

Normative transition rendering. Combining the parser's active envelope rules with the Cycle 1 artifact yields the modern conceptual form of a checkpoint transition:

Listing 5: Normative PACT v1 transition rendering for a checkpoint

```
{
  "p": "PACT",
  "v": 1,
  "type": "TRANSITION",
  "ns": "pact",
  "world": "open-field",
  "avatar": "wanderer-001",
  "action": "CHECKPOINT",
  "data": {
    "cycle": 1,
    "merkle_root": "sha256:
    d0082b508e01d45dee0dbccf6e8950dc039b4030c546aa475836b0008feccc71",
    "utterance_count": 4
  }
}
```

6 Merkle Tree Construction (RFC 6962)

6.1 Motivation

A protocol that anchors every individual claim on Bitcoin would be both expensive and slow. Each inscription occupies block space, pays fees, and competes with all other Bitcoin transactions for inclusion. A working Wanderer cycle may produce dozens of utterances; a busy year may produce thousands. Inscribing each one separately is not a realistic operating mode.

A naive alternative is to publish utterances off-chain on a server without any anchoring. This is cheap but provides no protection against later editing. A host that serves the worklog can silently modify, reorder, or remove entries; a verifier reading the worklog later has no way to detect the change. Off-chain logs without anchoring are convenient but not trustworthy.

A Merkle tree resolves this tension. The protocol commits a single hash, the Merkle root, on Bitcoin. The full set of utterances remains off-chain, where storage is cheap. From the Merkle root any verifier can later check whether a specific utterance belongs to the committed set. If a single byte of any utterance is altered, the recomputed root will not match the anchored one. Tampering becomes detectable without requiring all utterances to be re-fetched and compared individually.

The result is a deliberate balance:

- *economy* — only one root is anchored per cycle, independent of how many utterances the cycle contains,
- *accuracy* — every utterance is bound, byte-exact, to that root,
- *verifiability* — any third party can check membership of an utterance against the anchored root using only the utterance bytes and a small audit path.

Without this structure, PACT would have to choose between expensive universal anchoring and cheap unverifiable storage. The Merkle construction allows it to have both: many utterances per anchor, no loss of integrity per utterance.

6.2 Definition

PACT checkpointing uses an RFC 6962 style Merkle tree over canonical utterance bytes. The local checkpoint specification is explicit that this is domain-separated and append-oriented, not a duplicate-the-last-leaf construction.

Definition 2 (Canonical leaf hash)**.** *Given a canonical byte string u representing one utterance, its leaf hash is*

$$L(u) = \mathrm{SHA256}(0x00 \parallel u).$$

Definition 3 (Internal node hash)**.** *Given left and right child hashes x and y, each exactly 32 bytes, the parent hash is*

$$N(x, y) = \mathrm{SHA256}(0x01 \parallel x \parallel y).$$

6.3 Leaf hashing with domain separator 0x00

The leaf domain separator prevents ambiguity between raw utterance bytes and internal node concatenations. This is important because the protocol commits to byte strings, not to abstract trees independent of serialization.

The Cycle 1 tree file demonstrates that leaf hashes are stored as explicit `sha256:<hex>` strings after computation, but when computing higher nodes the underlying operation uses the raw 32-byte values.

6.4 Internal node hashing with domain separator 0x01

Using a distinct byte prefix for internal nodes ensures that no internal node hash can be confused with a leaf hash of some other byte string. This is the same structural safety idea used in Certificate Transparency [1].

6.5 Merkle root computation

For a non-empty ordered list of leaf hashes $h_1, \ldots, h_n$, define:

$$\mathrm{MTH}(h_1, \ldots, h_n) = \begin{cases} h_1 & \text{if } n = 1, \\ N(\mathrm{MTH}(h_1, \ldots, h_k), \mathrm{MTH}(h_{k+1}, \ldots, h_n)) & \text{if } n > 1, \end{cases}$$

where k is the largest power of two strictly less than n.
For the empty list, the checkpoint specification follows RFC 6962 and uses:

$$\mathrm{MTH}(\varnothing) = \mathrm{SHA256}(\epsilon).$$

6.6 Audit path verification

An audit path for a leaf consists of the sibling hashes needed to recompute the root from that leaf under the same split rule. Verification is successful if the recomputed root equals the advertised `merkle_root`. In operational terms:

1. canonicalize the target utterance,
2. compute its leaf hash with $0x00$,
3. combine with siblings using $0x01$ in the required left-right order,
4. compare the result to the root in the checkpoint artifact or on-chain transition.

7 State Transition

7.1 Motivation

In most database-backed systems, state is something you store. A table contains rows; the rows are the truth. To know what is true, you query the database. The system trusts whoever maintains it.

PACT inverts this picture. State is not stored as an authoritative object somewhere; it is *derived* by walking through Bitcoin block by block, validating each PACT envelope according to the same rules, and accumulating the result. Two parsers running the same rules over the same chain produce the same state. No party owns the state; no party can change it after the fact.

This matters for three reasons.

- *Reproducibility.* A claim about the current state of PACT can be checked by anyone with access to Bitcoin and the protocol rules. There is no privileged version of state held by a server. If a third party disagrees with what an explorer shows, they can re-derive state from the chain and verify the answer themselves.
- *Independence from indexers.* Block explorers and indexers are convenience layers. They speed up discovery, but they do not define truth. A misconfigured or malicious indexer cannot make a transition valid that PACT rules would reject, nor reject one that PACT rules would accept. State remains the same regardless of which indexer is consulted.
- *No need for a trusted custodian.* Because state is a function of chain plus rules, no organization needs to host state, maintain database integrity, or remain operational for state to be available. As long as Bitcoin persists and the protocol rules are known, state can always be re-derived.

The state transition function defined below is therefore not a description of how a database is updated. It is a description of how, given the same starting state, the same envelope, and the same chain context, every conforming implementation reaches the same next state.

7.2 Formal definition

Let S be parser state, e be a candidate PACT envelope, and C be the chain context that includes block ordering, witness bytes, and same-sat evidence. A state transition function can be written abstractly as

$$\delta : (S, e, C) \mapsto S'$$

subject to protocol validity predicates. More explicitly:

$$S' = \begin{cases} \text{apply}(S, e, C) & \text{if validEnvelope}(e) \wedge \text{validContext}(e, C), \\ S & \text{otherwise.} \end{cases}$$

This mirrors the operational behavior seen in the parser code: invalid payloads are rejected, and valid payloads are applied in canonical order.

7.3 Operational description

Operationally, the parser pipeline performs five conceptual stages:

1. fetch blocks and transactions from Bitcoin,
2. inspect witness data,

3. detect Ordinals-style envelopes,
4. decode and validate PACT payloads,
5. apply valid objects into state in deterministic order.

The order relation is induced by Bitcoin itself: block height first, then transaction index, and then the parser's object extraction order within a transaction when relevant.

7.4 Validity conditions

The active materials support the following high-level validity conditions:

- the envelope must parse as UTF-8 JSON,
- `p` must equal `PACT`,
- `v` must equal 1,
- the object type must be recognized,
- type-specific required fields must be present,
- transition semantics may require same-sat continuity or other world-specific rules,
- checkpoint artifacts must be byte-consistent with their underlying worklog and Merkle tree.

For Wanderer specifically, the checkpoint specification adds stricter canonicalization and closed-schema rules for `wanderer-utterance/v1` and `wanderer-checkpoint/v1`. Those are application-level schemas built on top of the more general PACT framing.

8 Verification

8.1 Browser-side verification

A central design choice of PACT is that verification does not require trusting any specific server, including the one that publishes the worklog. A reader with a modern web browser can verify a checkpoint independently, using only the published artifacts and standard browser tools. The procedure is:

1. download `utterances.jsonl` from the published location,
2. parse the utterance the reader wishes to verify,
3. canonicalize that utterance to its exact byte form,
4. compute its leaf hash using the browser's built-in SHA-256 function,
5. rebuild either the audit path or the full Merkle tree using the same hashing rules,
6. compare the resulting root to the root recorded in the on-chain checkpoint or the local `checkpoint.json`.

If the two roots match, the reader has confirmed three things at once: that the utterance was part of the committed cycle, that the worklog has not been altered since it was anchored, and that the checkpoint artifact is consistent with what is on Bitcoin.

The verification depends on byte-exact reproduction. The browser must hash the same canonical bytes that the original implementation hashed when it produced the checkpoint. If the browser reorders JSON keys, normalizes numeric formatting, or alters whitespace, the leaf hash will differ and verification will fail. This is not a flaw of the protocol but a property of any byte-committed system: small differences in serialization produce entirely different hashes. The canonicalization rules in the checkpoint specification are designed to remove this ambiguity.

Browser-side verification is the most accessible form of independent verification. It requires no software installation, no server, no special infrastructure. Anyone reading the published worklog can run the check themselves. This makes the protocol's guarantee of reproducibility practically available, not merely theoretical.

8.2 Independent parser verification

A parser is the program that reads Bitcoin transaction data and applies the PACT protocol rules to determine which inscriptions are valid PACT envelopes, which transitions belong to which avatars, and what the resulting state is. The local `pact-parser` repository provides one such implementation. Its documented model is:

```
Bitcoin -> Parsed PACT objects -> State -> StateHash
```

A parser must satisfy four properties to be considered reliable under PACT:

- *Determinism.* Given the same Bitcoin chain data and the same protocol rules, the parser must always produce the same output. Two runs of the same parser on the same input must yield identical state, identical acceptance decisions, and identical state hashes. Without this property, the parser's results cannot be compared or trusted.
- *Source independence.* The parser must derive its conclusions from raw Bitcoin block data, not from a third-party summary, label, or interpretation. If the parser depends on what some external service tells it about a transaction, then its conclusions inherit that service's biases, errors, or omissions.

- *Rule completeness.* The parser must apply the full set of PACT validity rules: envelope shape, type-specific field requirements, sat-bound continuity, and any world-specific rules that apply. Skipping a rule means the parser may accept invalid data or reject valid data.

- *Reproducibility across implementations.* A second parser, written independently in a different language by a different author, must produce the same output as the first when given the same input. This is what makes PACT a *protocol* rather than a single program: the rules are public, and any conforming implementation is as authoritative as any other.

Parser versus indexer. A parser and an indexer are not the same thing, and the distinction matters for verification. An indexer is a service that scans the chain, applies its own interpretation rules, and publishes the results in a queryable database. Block explorers, ordinals indexers, and other public services are indexers in this sense. They are convenient: they answer questions quickly without requiring the user to run any software.

A parser sits beneath the indexer in the trust stack. The indexer reports what it found; the parser determines, from first principles, what is actually there. When the indexer and the parser agree, the user can rely on the indexer for speed. When they disagree, the parser is authoritative, because it consulted Bitcoin directly and applied the protocol rules without intermediate interpretation.

A reliable parser, in other words, is what makes an indexer trustworthy. Without a parser, an indexer is a black box; the user must accept what it says without recourse. With a parser, the user has a check available: if the indexer's claim does not match the parser's conclusion, the user knows which to trust.

What independent verification means in practice. A user who wants to verify a specific PACT claim does not need to run their own parser for every check. Most users will rely on indexers and on the published cycle artifacts. But the option of running a parser must remain available, and the parser itself must be reliable in the four senses described above. If a second implementation produces the same acceptance decisions, the same tree roots, and the same state hashes on the same inputs, the protocol has achieved the kind of reproducibility it intends: truth that can be re-derived rather than truth that must be trusted.

8.3 Audit path verification walkthrough

The third form of independent verification is the most specific: checking whether a single utterance belongs to a committed cycle. This is what an *audit path* provides. An audit path is a short proof — typically a few hashes — that allows a verifier to reconstruct the Merkle root of an entire cycle starting from one leaf, without needing the full tree.

The mechanism is what makes the construction useful in practice. A cycle may contain hundreds or thousands of utterances. Asking a verifier to download the entire worklog and recompute the full tree to check a single claim is wasteful. The audit path lets the verifier check that one specific utterance is part of the cycle using only that utterance and a small set of sibling hashes — roughly the logarithm of the total number of leaves.

Why this works. The Merkle root of a cycle is computed by hashing pairs of leaves together, then pairs of those results, and so on, until a single hash remains. To verify that one leaf belongs to the root, the verifier needs only the sibling at each level of the tree. By hashing the leaf with its sibling, then the result with the next level's sibling, and so forth, the verifier reconstructs the path from the leaf to the root. If the reconstructed root matches the committed root, the leaf is provably part of the cycle. If even one byte of the leaf or one sibling has been altered, the reconstructed root will differ.

The procedure. Suppose a verifier wishes to check whether a specific Cycle 1 utterance belongs to the committed Merkle root. The steps are:

1. Read the exact line from `utterances.jsonl`. The line must be retrieved without modification — no whitespace changes, no key reordering, no character normalization.

2. Parse the line as JSON and canonicalize it according to the rules in the checkpoint specification. Canonicalization ensures the verifier hashes exactly the same byte sequence as the original implementation.

3. Compute the leaf hash $L(u) = \text{SHA256}(0x00 \parallel u)$, where u is the canonical byte string of the utterance and $0x00$ is the leaf domain separator.

4. Obtain the sibling sequence required to reach the root. This can come from a precomputed proof file, or the verifier can recompute the full tree from `tree.json` and read the siblings directly.

5. Combine the leaf hash with each sibling using the internal node rule $N(x, y) = \text{SHA256}(0x01 \parallel x \parallel y)$, respecting the left-right order at each level. Apply this combination level by level until only a single hash remains.

6. Compare the resulting hash with the `merkle_root` field in `checkpoint.json`. If they match, the utterance is confirmed as part of the committed cycle.

7. If an on-chain checkpoint transition exists for this cycle, compare the same root with the value carried in the inscription body. This step extends the verification beyond the off-chain checkpoint artifact to the Bitcoin chain itself.

What a successful verification proves. When all comparisons match, the verifier has established three facts simultaneously. First, the specific utterance was part of the cycle when the cycle was committed. Second, the worklog file has not been altered for this utterance since the cycle was sealed. Third, the off-chain checkpoint artifact is consistent with what is anchored on Bitcoin. Inclusion and consistency are proven together by the same calculation.

When verification fails, the failure points to one of three possible causes: the utterance was not in fact part of the committed cycle, the worklog has been modified after sealing, or the audit path is incorrect. The verifier cannot determine from the failure alone which of these caused it; further investigation is required. But the failure itself is reliable: a mismatch is not a coincidence, it is a signal that something has changed.

The cost of verification. For a cycle of n utterances, the audit path has length approximately $\log_2 n$. A cycle of one hundred utterances requires about seven sibling hashes; a cycle of one thousand requires about ten. Verification is therefore practical even for large cycles, and remains within reach of any device with a SHA-256 implementation. This is what makes byte-exact integrity checking accessible without specialized infrastructure.

9 Security Considerations

9.1 What PACT guarantees

Security in PACT means something different from what the term usually means in database-backed systems. In a conventional system, security is about controlling access: passwords, permissions, encryption of data at rest, audit trails. The operator of the database is the trusted party, and the security question is how to prevent unauthorized parties from reading or modifying what the operator has stored.

PACT has no operator. There is no database to protect, no central server to harden, no administrative account to compromise. The state of PACT is not held by anyone; it is derived from Bitcoin by anyone who runs the protocol rules. This changes the security question fundamentally.

In a database-less protocol, the security question becomes: what can be *relied upon* to remain true? A user who reads a PACT state today wants to know that the same state will be derivable tomorrow, next year, and a decade from now, regardless of who is still running services, regardless of which company is interested in the protocol, regardless of which servers remain online. The guarantee is not about confidentiality or access control. It is about *persistence of derivable truth.*

PACT can guarantee deterministic re-computability only within a specific scope. Subject to correct implementation and availability of the relevant Bitcoin data, PACT can provide:

- *Stable sat-bound identity tracking.* An identity constituted on a satoshi remains identifiable as long as Bitcoin records the satoshi's history. The identity cannot be revoked by an external party, cannot be claimed by a different sat, and does not require any service to remain operational. What the chain shows is what the identity is.
- *Byte-exact commitment of off-chain logs through Merkle roots.* Worklogs and other off-chain artifacts are bound to Bitcoin through cryptographic hashes. Any modification to the off-chain content, however small, is detectable through recomputation. The off-chain layer can move, mirror, or fail; the binding to Bitcoin remains.
- *Public reproducibility of parser decisions.* Any party with access to Bitcoin and the protocol rules can re-derive what is valid and what is not. There is no privileged interpreter whose conclusions must be trusted. Disagreements between implementations can be resolved by rerunning both against the same chain data.
- *Separation between transport and semantic interpretation.* Ordinals carry the bytes; PACT decides what they mean. A change to the transport layer does not change protocol meaning, and a change to interpretation does not require changing how bytes are carried. Each layer can evolve under its own rules without breaking the other.

These four guarantees together form the security model of PACT. None of them is about preventing access — the chain is public, and that publicness is part of the design. All four are about *durability of meaning*: that what was true yesterday remains derivable today, and what is true today remains derivable in the future, regardless of who maintains what.

9.2 What PACT does not guarantee

A protocol that claimed to guarantee everything would guarantee nothing reliably. PACT is deliberately scoped: it answers a specific kind of question, and stays silent on questions outside that scope. Understanding what PACT does not guarantee is therefore not a list of failures, but a description of the boundary inside which its guarantees apply.

Truth of content is not guaranteed. PACT does not guarantee that any claim written in a transition is semantically true in the world. If an avatar inscribes a transition that says *the source at this IRI states X*, PACT verifies that the inscription happened, that it was carried by the right sat, and that the transition is byte-exact as recorded. PACT does not verify that the source actually states X, nor that the source is reliable, nor that the avatar's interpretation is correct. The protocol guarantees that the claim was made; it does not guarantee that the claim is right.

This separation is intentional. A protocol that tried to adjudicate truth would need a trusted authority to do the adjudicating, and that authority would become a single point of control. PACT leaves truth-evaluation to those reading the record. The record itself is reliable; what readers conclude from it is their own work.

Discovery infrastructure is not guaranteed. Block explorers, ordinals indexers, mirrors, and aggregators are convenience layers. They make the chain easier to browse, but they are not part of the protocol. PACT does not guarantee that any specific explorer is complete, that any specific indexer is correctly configured, or that any specific mirror has the latest data. A user who relies entirely on one such service inherits that service's choices, biases, and failures.

The mitigation is direct: a user who needs certainty can re-derive state from Bitcoin using a reliable parser. Discovery services are useful for speed and convenience; they are not authoritative.

Key management is not guaranteed. PACT identity is bound to a satoshi, and that satoshi lives in a Bitcoin UTXO. Whoever controls the private key controlling that UTXO controls the identity. If the key is lost, the identity becomes inert — the satoshi can no longer be moved or used to issue further transitions. If the key is stolen, the new holder controls the identity as fully as the original holder did. PACT provides no mechanism for key recovery, no separate revocation channel, and no override.

This is a consequence of sat-bound identity working as designed. The same property that makes the identity uncensorable — that no external party can intervene — also means that no external party can help when something goes wrong with the key. Key management is the user's responsibility, and the protocol cannot substitute for it.

Social legitimacy is not guaranteed. A namespace, a world, or an avatar declared on chain has protocol-level existence, but that does not mean it is recognized, accepted, or considered legitimate by any community of practice. Anyone can declare a namespace named after an existing institution; the chain will record the declaration, but the institution is not bound by it. Anyone can claim to act on behalf of an organization through an avatar; PACT verifies that the claim was made, not that the organization endorsed it.

Legitimacy is a social matter, and PACT does not attempt to adjudicate it. The protocol provides a substrate on which communities can establish their own conventions of recognition. Those conventions are part of the world, not part of the protocol.

Compromise of off-chain infrastructure is not prevented. PACT anchors the integrity of off-chain artifacts through Merkle roots, but it does not prevent the off-chain infrastructure itself from being attacked, taken offline, or made inaccessible. A worklog server can be defaced, a mirror can disappear, a hosting provider can refuse service. What PACT guarantees is that any modification to the artifacts is detectable; it does not guarantee that the artifacts themselves remain available.

Users concerned about availability should mirror the artifacts they care about. The Merkle root anchored on Bitcoin remains verifiable as long as the underlying bytes are accessible somewhere — but somebody has to keep them somewhere.

Why these boundaries are appropriate. A protocol that tried to solve all of the above would not be a protocol; it would be a managed service with a service-level agreement, an operator, and a price. PACT is not that. It is a minimal substrate that does one thing well: it makes derivable truth durable. Everything else — semantic correctness, social legitimacy, key safety, server availability — is left to those who use the protocol, who must build their own conventions, infrastructure, and care for these matters.

The honest answer to *what does PACT guarantee* is therefore also the honest answer to *what does PACT not guarantee*: the protocol guarantees what can be derived from Bitcoin and nothing else. That is its strength. Anything more would require trust in something other than the chain, and that something would become the new point of failure.

9.3 Attack scenarios and mitigations

Attacks on PACT differ from attacks on conventional systems. In a database-backed system, an attacker typically aims to gain access, alter records, or impersonate a user. The defender's work is to prevent unauthorized actions: harden the perimeter, encrypt at rest, audit access, revoke compromised credentials. The system is in a defended state, and security is the work of maintaining that defense.

PACT does not have a defended state. The chain is public, the protocol rules are public, and anyone can read anything. There is no perimeter to harden, no access to control, no credentials to revoke. What can be attacked, then, is not the system itself but the *interpretation* of the system: making readers believe something that is not derivable from the chain.

This is a different kind of attack surface. It is unfamiliar to those used to thinking about firewalls and authentication. But it is well-defined, and each known attack pattern has a corresponding mitigation that requires no special infrastructure — only the discipline of returning to the chain when in doubt.

The attacks below are not exhaustive. They are the patterns most likely to arise in practice, given how PACT is designed to be used. Each is a way of producing a false belief about what is on chain; each mitigation is a way of bringing the question back to what the chain itself shows.

Indexer disagreement. Two block explorers or ordinals indexers may report different results for the same chain data. One may show a transition as valid that another shows as invalid. One may attribute an avatar to a sat that another attributes to a different sat. A user relying on a single indexer has no way to know which is correct — or whether either is correct.

This is not necessarily malicious. Indexers can disagree because they apply different rules, run different code versions, or have incomplete chain data. But the effect on the user is the same: two sources of truth, and no way to choose between them from the indexers alone.

Mitigation. Parse raw Bitcoin data using a reliable parser and apply the deterministic protocol rules locally. The parser's conclusion is authoritative because it derives from the chain directly, not from any intermediate interpretation. When indexers disagree, the parser is the tiebreaker.

Tampered off-chain worklog. PACT anchors a Merkle root for each cycle, but the underlying worklog (the JSONL file containing the actual utterances) lives off chain. A malicious or compromised host could serve a modified version of the worklog: deleted entries, reordered entries, edited text. Without verification, a reader downloading the worklog has no immediate way to detect the change.

This attack is particularly concerning because the modification can be subtle. Changing a single word in one utterance is enough to shift its meaning, but visually the worklog appears intact. Without recomputing the cryptographic binding, the alteration is invisible.

Mitigation. Compute the byte hash of the worklog file and compare it to the `data_commitment` field in the checkpoint. Independently, recompute the Merkle root from the worklog and compare

it to the root anchored on Bitcoin. If either comparison fails, the worklog has been altered. The attack is detectable; it cannot succeed against a verifying reader.

Semantic confusion between historical and active schemas. PACT has evolved. Early Bitronaut payloads used machine-centric fields and a different envelope structure. The active PACT v1 envelope uses uppercase `PACT`, avatar-centric fields, and a strict type set. Both kinds of inscriptions exist on chain.

An attacker — or simply a careless reader — could present a historical payload as if it were active PACT v1, or apply v1 rules to historical content and conclude wrongly that the historical content is invalid PACT data. Either confusion produces false beliefs about what the chain currently records under the active protocol.

Mitigation. Enforce uppercase `PACT` v1 envelope rules from the active parser. Treat earlier payloads as historical predecessors with their own context, not as defective v1 inscriptions. The parser should clearly distinguish between *not v1* and *invalid v1*; conflating these creates exactly the confusion the attack exploits.

False confidence from intermediate roots without archival. A cycle may publish intermediate Merkle roots before its final checkpoint is anchored on Bitcoin. These intermediates can be useful for transparency: they let readers see that work is in progress and that earlier states have been committed to. But intermediates carry trust only as long as they are independently preserved. If they are kept only on the original publisher's server, the publisher can later modify them without leaving trace.

A reader who relies on intermediate roots without verifying that they have been archived elsewhere may be deceived twice: first by trusting the intermediate as a commitment, second by discovering — too late — that the intermediate was rewritten.

Mitigation. Mirror or archive intermediate roots independently of the original publisher. Standard archiving infrastructure (Internet Archive, third-party mirrors, signed backups) is sufficient. Once an intermediate is archived in a location the publisher does not control, it can serve as tamper-evidence; before that, it cannot.

False same-identity claims. A new inscription may claim to be a continuation of an existing avatar without actually being inscribed on the same sat. The inscription text may include the avatar's name, reference its constituting inscription, and use the right vocabulary — yet land on a different sat than the original. To a casual reader, the inscription looks like a legitimate transition; to a verifier, it is a separate identity making unauthorized claims.

This attack works because identity in human reading is partly a matter of name and context. A reader sees the name *wanderer-001* and assumes the inscription is part of Wanderer's history. The protocol does not assume this; it checks.

Mitigation. Verify same-sat identity by following the satoshi through the Bitcoin transaction flow from the original constituting inscription forward. Do not trust an inscription's self-description as proof of identity. If the inscription is not on the same sat, it is a different identity — regardless of what its text says.

The pattern across all attacks. Each of the attacks above shares a common structure: an attacker or a careless source presents a derivation that does not actually follow from the chain, hoping the reader will accept it without re-checking. Each mitigation has the same shape: return to the chain, apply the protocol rules, and verify directly.

This pattern suggests a discipline for users of PACT. Rely on indexers, explorers, and mirrors for convenience. Treat their output as provisional. When something matters — when a decision or a public claim depends on it — re-derive from the chain. The extra step is the cost of operating without a trusted custodian; it is also why the protocol can offer durable truth in the first place.

10 A working demonstration: OrdinalsWall

10.1 What OrdinalsWall is

OrdinalsWall is a working interface for the PACT ontology as realised through Ordinals on Bitcoin. It makes that realisation readable and verifiable. It is not a platform, a service, or an authoritative index. It is a small set of static HTML pages, served from a regular web server, that fetches the public parser-state and renders it client-side. The pages contain inline JavaScript only for rendering and verification; no build-time generator stands between the chain and the reader. The site is available at ordinalswall.com.

Where the specification describes the ontology and its current v1 realisation, OrdinalsWall shows what that realisation produces for a reader. Where the specification defines verification, OrdinalsWall performs it in the browser. The site is therefore not a separate work next to the spec; it is the spec made visible and operable.

10.2 The structure of the site

OrdinalsWall is organised into four main views, each addressing a different aspect of the PACT ontology as it is presently realised on chain through Ordinals. A reader can navigate among them through a top navigation bar, and any object referenced in any view leads to the same Inspector page where its raw form and chain context become visible. A related interface, the Wanderer cycle ledger (wanderer.bitronaut.io/cycles.html), shows the same client-side rendering pattern applied to a single avatar's cycle history.

Current. The Current view shows what is presently accepted as conformant to PACT v1. Namespaces, worlds, avatars, and transitions that satisfy the active envelope and validity rules described in Sections 4 and 5 appear here. When the v1 layer contains no objects yet, the page states this honestly rather than concealing the absence; the reader is directed to the historical layer for what already exists on chain. The Current view is therefore not a status dashboard. It is an answer to the question: what does the protocol recognise as authoritative right now?

Historical. The Historical view shows what the parser recognises as predecessor material under the rules of Section 5.1. These inscriptions are part of the chain and remain readable, but they do not satisfy the v1 envelope. The view groups them by the reason they are non-normative (`historical-flat`, `historical-constitution`, `historical-machine-centric`) and shows the cause for each (`lowercase-p`, `machine-field`, etc.). The Historical view exists because forgetting predecessors would falsify the chain's history. The protocol evolves; what came before remains visible.

Graph. The Graph view shows the hierarchical relationships between object types. Namespaces contain worlds, worlds contain avatars, avatars produce transitions. Where Current and Historical present objects as flat lists, Graph presents them as the ontology described in Section 3: a structure in which each object has its place and each relation is navigable. The Graph view exists because PACT is not a collection of records but a system in which meaning emerges from relations. Reading the chain as a list misses what the chain actually contains.

About. The About view contains the original prose introduction to OrdinalsWall, retained verbatim from the site's earliest public version. It addresses the reader directly: what is this place, why does it exist, what is it trying to demonstrate? Where the other three views show data, About provides the framing in which the data acquires meaning. The About view exists because a working demonstration without context is just an interface; with context it becomes an argument.

Inspector. Beyond the four main views, every object referenced anywhere on the site links to a single detail page that shows its raw payload, chain identifiers, and verification controls. This page is not a tab in its own right; it is the destination at which all paths through the site converge. The Inspector is where the protocol's claim of independent verifiability becomes operational: a reader can read the bytes, follow the inscription to ordinals.com or mempool.space, and for checkpoint objects perform the Merkle reconstruction in the browser.

The four-view structure mirrors the way the specification itself is organised. Current and Historical correspond to what the protocol recognises now and what it acknowledges from before. Graph corresponds to the ontological structure that Sections 3 and 4 define. About provides the orientation that any reader needs before the technical sections become usable. Together, the views are not features added to a site; they are the spec made navigable.

10.3 Verifiable references throughout the site

Every claim made on OrdinalsWall is anchored to verifiable chain data through links and hash references. The reader is not asked to trust what the site shows; the reader is given paths to confirm it independently. This subsection lists which references appear on which view and what each one makes verifiable.

Inscription identifiers. Every PACT object visible on the site carries its inscription identifier, the canonical reference under Ordinals. The identifier links to ordinals.com/inscription/`<id>`, where the raw inscription content can be read directly from chain data. This link appears on Current, Historical, Graph, and Inspector. It is the single most important anchor on the site: any object whose inscription identifier resolves to expected content has been verified at the transport layer described in Section 2.4. An example: the Wanderer-001 constituting inscription is `1266de0c...9e40i0`.

Transaction identifiers. For each inscription the site also exposes its underlying transaction identifier, derived by removing the trailing `i0` from the inscription identifier. The transaction links to mempool.space/tx/`<txid>`, where block context, fee, witness data, and confirmation status can be inspected. This link appears on Historical and Inspector. A reader can confirm not only that an inscription exists, but at what block it was confirmed and under what conditions. An example: the cycle 16 checkpoint reveal transaction `eb5468f3...1b70` can be inspected on mempool.space.

Satoshi identity links. For sat-bound objects, principally avatars, the carrier satoshi appears prominently and links to ordinals.com/sat/`<sat>`. This is not a metadata field; it is the identity anchor itself, as defined in Section 2.3. The sat link appears on Current (for accepted avatars), on Historical (where sat data is available), on Graph (where avatars appear as nodes), and on Inspector. Following the link reveals the satoshi's full inscription history and current location in the UTXO graph, the basis for same-sat continuity verification. An example: the Wanderer-001 carrier sat 1168224522821204 can be inspected directly.

Block references. Block heights are shown alongside every object as the chronological anchor. Where space permits, block heights link to mempool.space/block/`<height>`, exposing the block in which the inscription was confirmed and its position in Bitcoin's append-only history. Block references appear on all four main views and on Inspector. An example: the block that first anchored the Wanderer-001 constitution, 946186, can be inspected directly.

Hash fields as visible anchors. For checkpoint objects, three hash fields appear in the Inspector view: `merkle_root`, `data_commitment`, and where applicable `tree_commitment`. These

are not links but visible bytes that the reader can copy and verify independently. The merkle root corresponds to the RFC 6962 reconstruction defined in Section 6; the data commitment binds the off-chain worklog to a fixed byte sequence; the tree commitment binds the auxiliary tree artifact. All three appear in raw hex form so that no interpretation by the site stands between the reader and the bytes. An example: the cycle 16 checkpoint `eb5468f3...1b70i0` shows these fields in its Inspector view.

Internal references. Every object on Current, Historical, and Graph links to its own Inspector page through an internal reference of the form `/object.html?id=<inscription_id>`. The Inspector is where all paths converge: the raw payload appears in full, the chain identifiers are listed, the verification controls become available. The internal reference is not external verification, but it is the gateway to it. An example: the cycle 16 checkpoint can be opened through `/object.html?id=eb5468f3...1b70i0`.

What the references do not do. External links open external services. ordinals.com is an indexer; mempool.space is a block explorer. The site treats both as conveniences, not authorities. A reader who distrusts an external explorer can run their own parser against Bitcoin and reach the same conclusions the site shows. The links are offered because they are useful, not because they are required for verification.

The pattern across all references is the same: every claim points to data the reader can fetch independently. The site does not hold authority; it holds invitations to verify.

10.4 How the site embodies the specification

Three aspects of the site correspond directly to sections of the specification:

- The four object types defined in Section 4 (`NAMESPACE`, `WORLD`, `AVATAR`, `TRANSITION`) appear in the site as both a hierarchical navigation graph and as individual inspection pages.
- The Merkle tree construction defined in Section 6 is not only described in the site's checkpoint inspection pages but actually performed there. When a reader opens a checkpoint object and clicks verify, the browser reconstructs the RFC 6962 Merkle root from the utterance bytes using the leaf prefix $0x00$ and internal node prefix $0x01$, exactly as defined in Section 6. The result is compared with the root anchored on Bitcoin. Verification happens locally, in the reader's browser, with no privileged service involved.
- The three forms of verification described in Section 8 are all represented. Browser-side verification of cycle artifacts is implemented directly. Independent parser verification is supported by the public availability of `parser-state.json`, which any reader can read and re-derive from chain data. Audit path verification is supported through the Merkle reconstruction described above.

10.5 What the site does not claim to be

OrdinalsWall is not the authoritative source of PACT state. The authoritative record is the Bitcoin chain as made individually readable through Ordinals. The site reads what the parser has derived from that record and presents it. If the site were to disappear tomorrow, every claim it shows would remain derivable from chain data alone. This is not a weakness of the site; it is the property the protocol is designed to guarantee.

The site is also not exhaustive. It shows what the parser has accepted as conformant to PACT v1, what it has recognised as historical predecessor, and what it has rejected as outside the protocol entirely. A reader who wishes to investigate further can follow inscription identifiers to external explorers or run an independent parser against Bitcoin directly.

10.6 Avatar perspective: cycles as cognition in time

The site's other views show the protocol from a horizontal perspective: namespaces, worlds, avatars, and transitions as four equal categories. The Avatar view introduces a different reading angle. It places one avatar at the centre and reads its presence on chain as a sequence of cycles, each cycle a sealed period during which the avatar received, processed, and committed observations.

For the present implementation, the only avatar with a cycle history is Wanderer-001, constituted on satoshi 1168224522821204 at block 946186. The Avatar view shows Wanderer's cycles in chronological order, each cycle presented with its cryptographic anchors (merkle root, data commitment, tree commitment), the inscription that previously checkpointed its lineage, and the satoshi to which its identity is bound. Each cycle is followed by its utterances: short, sourced statements organised by zone (forehead, jaw, temple, and so on, following the conventions described in Section 7.3), each carrying a confidence score and a cited source whose hash is included so the reader can verify what the avatar was looking at.

Where Current and Historical answer the question *what does the protocol contain?*, and Graph answers *how do objects relate?*, the Avatar view answers *what does an entity on chain do over time?*. This last question matters because PACT is not only an ontology; it is a protocol for cognition that leaves traces. An avatar is not just a node in a graph but a presence whose actions become part of Bitcoin's history. Reading the chain as a sequence of avatar cycles is one way of recovering that presence.

The view follows the design pattern of the related interface at wanderer.bitronaut.io/cycles.html, which presents the same kind of avatar narrative for Wanderer-001 alone. OrdinalsWall's Avatar view inherits that pattern and prepares for future avatars: when a new avatar is constituted on its own carrier sat, the view will accommodate it without architectural change. Identity is sat-bound; the view follows the sats.

Verification on the Avatar view is identical to that on the Inspector. Every cycle exposes a verify control that reconstructs the RFC 6962 Merkle root from local utterance bytes and compares it with the root anchored on Bitcoin. A reader who wants to confirm what the avatar has said does not need to trust the site; the bytes and the algorithm are both inspectable.

10.7 Historical predecessors as visible heritage

The site implements the three-category classification described in Sections 5.1 and 9.3. Accepted objects are shown as the current PACT v1 layer. Historical predecessors are shown in their own view, grouped by sub-type, with the reason for their non-normative status visible (`lowercase-p`, `machine-field`, etc.). Rejected payloads, including unrelated protocols and malformed data, are reported as aggregates only.

The historical layer is not hidden. It is presented as part of what the chain contains, recognisable but not normative. This corresponds to the protocol's stance that earlier schemas are part of PACT's evolution and remain visible on Bitcoin regardless of subsequent versioning. A reader sees the open-field constitution as it was inscribed, the cycle 16 checkpoint (`eb5468f3...1b70i0`) in its original lowercase form, and the Wanderer-001 constitution (`1266de0c...9e40i0`, block 946186, on sat 1168224522821204) on its carrier sat. Nothing is rewritten; everything is read.

11 Intellectual context

PACT does not arise from nothing. It draws on three intellectual traditions that have shaped how computing systems represent knowledge, structure programs, and persist data. Each tradition contributes a distinct insight to PACT's design, and the combination produces something that none of them on its own would have produced.

11.1 Ontology versus database

The most important distinction underlying PACT is the difference between an ontology and a database. Both are ways of organizing information, but they answer different questions and behave in fundamentally different ways.

A database stores rows. The rows are the truth: to know what is true, you query the database, and the database returns what its operator has stored. The operator's authority is implicit in every answer. If the operator decides to change a row, the truth changes. If the operator decides to delete a row, the truth disappears. A database is, in this sense, a set of facts held in trust by whoever runs the database.

An ontology is something different. An ontology is a formal description of *what kinds of things exist* in a domain, *how they relate to one another*, and *what rules govern their behavior*. An ontology does not store data; it defines the structure within which data has meaning. Two implementations of the same ontology, given the same data, must arrive at the same conclusions, because the ontology specifies what conclusions are valid.

The distinction matters because it determines what kind of system you can build. A database-backed system requires a trusted custodian to hold the data. An ontology-based system requires no custodian: anyone who knows the ontology and has access to the data can derive the same conclusions independently. The first model centralizes knowledge in an authority; the second model distributes the capacity to reason.

11.2 Why this matters for Bitcoin Ordinals

Bitcoin provides the permanent substrate; Ordinals makes that substrate usable for inscriptions by attaching arbitrary bytes to individually addressable satoshis. The substrate is public, persistent, and ownerless — no one party maintains it, no one party can revoke or alter what is written. This is the opposite of a database environment. There is no operator to trust, no custodian to query, no service to depend on.

A database approach to Ordinals would require building a custodial layer on top: a centralized indexer that interprets inscriptions and serves answers to queries. This is what most ordinals-based services do. They are databases over Bitcoin: their answers are only as reliable as the indexer that produces them, and their state can be lost, censored, or rewritten if the indexer fails or is compromised.

PACT takes the ontology approach instead. The protocol defines what kinds of objects exist (namespaces, worlds, avatars, transitions), how they relate (sat-bound continuity, world context, transition semantics), and what rules govern their validity (envelope structure, type-specific requirements, canonicalization). Given these definitions and access to the chain, anyone can reason directly about what is on Ordinals. No custodian is needed. The protocol is the ontology; Bitcoin is the permanence layer; Ordinals exposes the addressable inscription surface; the conclusions are derivable by anyone willing to do the work.

This is what makes direct reasoning on an Ordinals environment possible. Without an ontology, Ordinals is just a stream of bytes — each inscription self-describing, each interpretation ad-hoc. With an ontology, Ordinals becomes a structured space in which questions can be asked and answered with confidence: who is this avatar, what world does this transition belong to, is this checkpoint valid? The answers are not stored anywhere; they are derived.

11.3 Why ontology-based systems are difficult to build

Ontology-based systems are powerful, but they are harder to construct than database-backed systems. The difficulty is mostly intellectual rather than technical.

A database can be designed incrementally. You start with a few tables, add more as needed, change the schema when requirements change, and the operator absorbs the consequences of those changes. The system tolerates inconsistency because the operator mediates between users and data.

An ontology cannot work this way. If the ontology changes, every piece of data interpreted under the old version remains interpretable, but every implementation must be aware of the change. There is no operator to mediate; the rules themselves are the mediation. This forces the designer to think carefully upfront: what really exists, what really relates to what, what rules really hold. Mistakes in the ontology propagate everywhere and cannot be silently corrected.

This is why companies that have invested heavily in ontology — Palantir is the best-known example — emphasize ontology design as their core differentiator. Their products organize institutional data not by storing it in conventional tables, but by defining ontologies under which the data acquires meaning. Their clients pay substantial sums for this work because the result is a system that can answer questions across data sources that share an ontology, in ways that ordinary databases cannot.

What Palantir keeps proprietary, however, is precisely what PACT makes public. Palantir's ontologies live inside Palantir's software and serve Palantir's clients. The clients trust Palantir to maintain the ontology correctly; they accept the custodial relationship as the price of the capability. PACT's proposition is different: the ontology is the protocol, the protocol is open, and anyone can reason under it without paying for access to the custodian. Bitcoin Ordinals provides the permanent substrate; PACT provides the ontology that makes the substrate intelligible.

The two approaches are not in opposition. Palantir's model works for organizations that have a custodian and want one. PACT is for contexts where no custodian is desired, where the durability of derivable truth matters more than the conveniences of a managed service, and where the substrate must outlive any individual organization.

11.4 The traditions PACT draws on

Three intellectual traditions converge in PACT.

The Semantic Web. Tim Berners-Lee proposed the Semantic Web as an extension of the World Wide Web in which information has explicit meaning, machines can reason about it, and knowledge can be shared across applications without prior coordination. RDF (Resource Description Framework) [5] and OWL (Web Ontology Language) are the technical artifacts of this tradition; the larger contribution is the idea that knowledge can be expressed in a form that is both machine-readable and meaningful. The world `open-field` within `pact-semantics` is a direct application of this tradition: avatars observing Wikidata entries, anchoring their observations in PACT, and producing a record that can be reasoned about across implementations.

Frank van Harmelen, professor of Knowledge Representation and Reasoning at the Vrije Universiteit Amsterdam, has shaped how ontologies are taught and applied in practice. His work on large-scale semantic-web systems and on the relationship between formal logic and real-world data informs how PACT thinks about worlds: not as arbitrary collections of inscriptions, but as contexts with declared rules under which avatar transitions acquire meaning.

Engineering minimalism. Andrew Tanenbaum's work on operating systems, especially MINIX, articulated a principle that runs throughout PACT: systems should be small, clean, safe, and understandable. Large systems hide complexity in ways that make verification impossible; small systems make verification trivial. PACT's minimal envelope, limited type set, and refusal to add

features without justified need follow this principle directly. A protocol that can be read in a single afternoon and implemented in a single weekend is more likely to be correctly understood and reliably reproduced than a protocol that requires extensive tooling to interpret.

Bitcoin and Ordinals. Bitcoin, designed by Satoshi Nakamoto [2], provides the persistent ordering substrate without which PACT would have nothing to anchor to. Casey Rodarmor's [3] playful Ordinals protocol makes Bitcoin's permanence individually addressable and browser-reachable, and thereby makes sat-bound inscription practical: without ordinal numbering and the inscription envelope, satoshis would carry no payload and PACT would have no carrier. Blockamoto's framing [6] of Bitcoin Core as a habitable world rather than a narrow payments substrate provided the conceptual opening in which a protocol like PACT could be imagined at all. Peter Todd's OpenTimestamps [4] demonstrated, well before PACT, that Merkle batching against Bitcoin is a workable pattern for durable timestamping; PACT's checkpoint protocol is a direct descendant of this idea, generalized from timestamps to identity-bound state transitions.

11.5 Beyond Ordinals: PACT as identity anchor

PACT is defined here as an ontology realised through Bitcoin Ordinals, but the principles it relies on extend further. The mechanism that makes a satoshi a durable identity anchor does not depend on Ordinals as a transport layer alone. What Ordinals supply is not only a Bitcoin-native way to attach payloads to outputs, but a way to make those payloads individually addressable and reachable through tooling that already exists; the deeper invariant is a globally ordered, owner-verifiable substrate on which an identity can accumulate attestations without a designated custodian. Another carrier that offered comparable durability and verifiability could in principle host an analogous envelope—Ordinals is simply the instantiation this specification standardizes.

The Semantic Web tradition already assumes that identity and meaning travel across systems: URIs denote resources, vocabularies declare interpretation rules, and agents apply those rules to produce derivable conclusions. PACT aligns with that picture when worlds such as `open-field` anchor observations about Semantic Web sources on chain; the Bitcoin layer does not replace RDF or OWL but grounds who accepted what, under which declared rules. Chain-backed identity thus becomes an anchor for statements whose logical shape remains Web-standard even though their custody pattern does not.

HTTP is the ambient protocol for retrieval on today's Internet. Nothing in PACT requires HTTP for consensus or verification—canonical truth remains whatever validators derive from Bitcoin and inscribed bytes—yet HTTP is the natural exchange surface when implementations fetch evidence, dereference vocabulary URLs, or coordinate services. In that role, PACT can serve as an identity anchor for HTTP-based communication: an avatar or namespace established through chain evidence supplies a stable, cryptographically constrained party identifier for requests, responses, and linked data flows that need not be expressed as Ordinal inscriptions themselves. This specification deliberately stops short of normative bindings to particular HTTP profiles or discovery mechanisms; those are applications of the same commitments formalized here.

11.6 What this combination produces

These three traditions, brought together, produce something that none of them on its own would have produced. The Semantic Web tradition gives PACT its concept of meaning across implementations. Engineering minimalism gives PACT its discipline of staying small. Bitcoin gives PACT's present realisation its permanence and ownerlessness; Ordinals makes that permanence individually addressable, inspectable, and available to ordinary readers.

The result is a protocol that is at once an ontology (in the Semantic Web sense), a small system (in the Tanenbaum sense), and a Bitcoin-anchored construct (in the Ordinals sense). It is none of these alone; it is what happens when all three are taken seriously together, and used

to address a question that none of them alone could fully answer: how to make derivable truth durable across time without depending on any organization to maintain it.

12 Adoption, usability, and portability

PACT v1 defines a concrete realisation of the PACT ontology for identity and anchored action through Ordinals on Bitcoin. The previous chapters explained what the protocol contains, how it verifies, and where it draws its intellectual lineage. This chapter addresses a different set of questions: who can build under PACT, why its structure may feel familiar to those coming from software engineering, and what makes it portable across implementations. These questions matter because a protocol that no one else can build under is, in practice, not a protocol but a personal project. PACT is intended as the former.

12.1 Adoption: who can build under PACT

The protocol does not enforce a registry of permitted namespaces. Anyone can declare a namespace by inscribing a NAMESPACE object through Ordinals on their carrier sat, with Bitcoin supplying the permanence on which that declaration remains visible. There is no application process, no central registrar, no body that grants permission. The chain is the arbiter; what is on chain exists.

This invites two questions. First: what stops conflict? Two parties could in principle both claim the same namespace name. The answer is that namespaces are not trademarks. The earlier inscription has earlier provenance, and any later parser can recognise that. But more importantly, namespaces are intended as invitations rather than claims. A namespace named `commerce` does not belong to whoever inscribed it first; it offers a vocabulary that others may extend under, conflict with, or ignore. The chain records all positions; readers and parsers decide what to recognise.

Second: what would a second adopter actually do? Concretely: choose a sat as their identity anchor, inscribe a NAMESPACE object declaring the name they intend to use, define one or more WORLD objects under that namespace specifying the rules and vocabularies they expect to apply, and begin inscribing AVATAR and TRANSITION objects within those worlds. The PACT v1 envelope rules apply uniformly. A second adopter does not need permission from the first. The two namespaces sit in the same protocol, on the same chain, available to the same readers.

The current state of adoption is honest to acknowledge. At the time of writing, the only namespace formally declared under PACT v1 is `pact-semantics`, with its world `open-field`. The protocol's demonstration is therefore a single working example. This is not unusual: most protocols begin with one adopter. What matters is whether the structure permits others to follow, and whether the cost of doing so is low enough that they will.

12.2 Usability: a familiar object-oriented form

A reader with a background in object-oriented programming may notice that PACT's four object types have a structure that feels familiar. The same shape that organises Java packages, modules, instances, and method invocations appears here in the context of Ordinals inscriptions anchored on Bitcoin, with some adjustments worth making explicit:

- A `NAMESPACE` resembles a Java package: a name-bearing container for related definitions, with no executable behaviour of its own. Several worlds may live under the same namespace, just as several classes may live under the same package.
- A `WORLD` is closer to a module or singleton than a class. A world is not a template that produces instances; it is a uniquely defined set of rules and vocabularies, inscribed once on chain, that governs what may happen within it. Inscribing a second world with the same name under the same namespace would be a conflict, not a second instance.
- An `AVATAR` is where instances actually appear. Multiple avatars may exist within the same world, each bound to its own carrier satoshi and each carrying its own history of transitions.

Two avatars in `open-field` share the rules of that world but are otherwise independent entities.

- A `TRANSITION` resembles a method invocation: an action performed by an avatar in accordance with the world's rules, recorded with arguments (the `data` field) and a target (the world).

The analogy is useful for orientation but not exact at every point. Java packages exist only at compile and runtime; PACT namespaces persist on chain indefinitely. Java modules can be redefined or replaced; a PACT world inscription cannot be modified once on chain. Java instances are typically transient and may be garbage collected; PACT avatars are sat-bound and unforgeable for as long as Bitcoin itself runs. The analogy describes the shape of the ontology, not the lifetime or modification model.

What the analogy does capture well is the relationship between the four types. A namespace organises; a world defines; an avatar exists within and acts; a transition records the action. This is the same structure a developer encounters when reading any modular software system: containers organise definitions, definitions specify behaviour, instances do work, and method calls record what work was done. PACT places this structure on a substrate that no implementation owns, and that gives it some properties software cannot have. But the mental model required to read PACT is not a new mental model.

Why this matters: a developer encountering PACT for the first time does not face an entirely novel structure. The mental model required to read a PACT inscription, decide which envelope fields belong to which type, and understand how transitions act within worlds, is the model already used to reason about modular software. The protocol's accessibility benefits from this familiarity.

12.3 Portability: a Model-View-Controller separation

PACT achieves a separation of concerns that maps cleanly onto the Model-View-Controller pattern. The Model is the chain record as exposed through Ordinals: every PACT inscription is authoritative data, recorded permanently on Bitcoin and made individually readable without privilege. The Controller is the parser: it applies the envelope rules, validates conformance, classifies historical predecessors, and produces a state that is logically a function of the chain. The View is any interface that reads parser output and presents it to a human, machine, or further protocol layer.

The point of MVC is that the three layers are independent. A different View can present the same Model through a different parser, or even with a different organising principle. A different Controller can validate the same chain inscriptions under different rules, perhaps with a future PACT v2 envelope. The Model is the shared invariant; the Controller is the interpretation; the View is the presentation.

This independence is what makes PACT portable. The chain is not owned by any implementation. The parser can be rewritten in any language, by any author, against the same canonical envelope rules. The interface can be a website, a command-line tool, an API endpoint, or a future medium not yet imagined. None of these depends on the others continuing to exist; each can be replaced without losing what is on chain.

The pattern also has its limits. MVC was designed for desktop GUIs in the 1970s, where the View was a window the user interacted with directly. In PACT, the *primary* interface for many use cases will not be human; it will be machine. An AI agent reading parser output is the View in the same sense that a human looking at a webpage is. The pattern survives the change of medium, but the metaphor stretches.

12.4 What this combination produces

Adoption, usability, and portability are not independent properties. A protocol that is portable but unusable will see no adoption. A protocol that is adopted but locked to one implementation is not really portable. PACT's structure is intended to make all three reinforce one another: the form is familiar enough to lower the cost of adoption, the specification is small enough to permit independent implementation, and the chain, as made addressable through Ordinals, prevents any single implementation from becoming authoritative over what has been recorded.

Whether this combination produces wide adoption is a question only time can answer. What can be said now is that the structure does not actively prevent it. A second adopter does not need to ask permission. A second implementation does not need to coordinate with the first. A second view of the chain can present PACT objects in any way that respects what is there. The protocol succeeds, in this sense, when others build under it without being asked.

12.5 Computation as a future protocol layer

PACT v1 defines identity, world membership, and the recording of avatar actions on chain. What it does not define is computation. An avatar can declare, observe, and checkpoint, but the protocol itself takes no position on what it means for an avatar to compute – to take inputs, apply an algorithm, and produce a verifiable output that others can reproduce.

This is not an oversight. It reflects a deliberate choice to keep PACT v1 small. Identity is the foundation; everything else builds on top. Computation is one of the things that could build on top, and the form it would take is worth sketching, even though its full definition belongs to a separate document.

A computation protocol on PACT would treat verifiable computation as a kind of transition. An avatar would inscribe a TRANSITION whose action declares that a specific algorithm, identified by its own inscription, was applied to a specific input, identified by its hash, producing a specific output, also identified by its hash. Any reader could fetch the algorithm, run it against the input, and verify that the output matches. The result is a chain record of what was computed, by whom, with what inputs, producing what – without trusting that any particular party performed the computation correctly. Bitcoin holds the claim durably; Ordinals makes the claim reachable to the reader, who confirms it.

This pattern would not require PACT v1 to change. Computation transitions would simply be transitions whose action is COMPUTE (or whichever name the future protocol chooses) and whose data fields follow the new protocol's rules. Existing parsers would recognise them as transitions of an unfamiliar kind and pass them through unchanged. Parsers built for the new protocol would interpret them more deeply, fetching algorithms and verifying outputs. The permanence substrate is shared and the addressable inscription surface is shared; the interpretation differs.

Several precedents suggest this direction is workable. RFC 6962 already provides verifiable Merkle construction, and OrdinalsWall already uses it to verify checkpoint roots in the browser. Reproducible builds, as practised in the open-source community, already establish that a deterministic algorithm applied to a fixed input always produces the same output. The combination – inscribe the algorithm, inscribe the input, inscribe the claimed output, let readers verify – requires no new mathematics, only a new envelope.

What such a protocol would not be is a complete programming language. A computation protocol records that a computation occurred and provides the data to reproduce it. It does not specify a syntax, a type system, or an evaluation semantics for arbitrary programs. Those questions belong to language design, which is a longer endeavour. A protocol is smaller and reaches usefulness sooner.

The natural place for such work is under a separate namespace. A future author might declare a namespace named `compute` or `verifiable-execution` and design its envelope rules without consulting any existing PACT v1 author. The chain would record both PACT v1 inscriptions

and the new namespace's inscriptions side by side. Each parser would recognise what it was built to recognise. No coordination is required; the protocols would co-exist.

PACT v1 is therefore not the end of what can be done on this foundation. It is the part that must exist first: identity before action, action before computation. When computation arrives, it will not arrive as a replacement but as a layer that takes identity for granted and adds the rest. That layer is left for those who want to build it.

12.6 Financial transactions as a parallel example

Computation is one direction PACT could grow in. Another, quite different, is the coordination of financial actions on Bitcoin itself. The pattern that PACT establishes – identity bound to a sat, world membership, transitions as recorded actions – maps surprisingly well onto the process of constructing, signing, and broadcasting Bitcoin transactions among multiple parties.

Consider a hypothetical namespace `financial-transactions` with a world named `psbt`. The world would govern transitions corresponding to the lifecycle of a Partially Signed Bitcoin Transaction, defined in BIP 174: an avatar inscribes `PSBT_INITIATE` to declare the start of a transaction, including its inputs and outputs; other avatars inscribe `PSBT_SIGN` as they add their partial signatures; one avatar inscribes `PSBT_FINALIZE` when all required signatures are present; and `PSBT_BROADCAST` records the moment the transaction enters the Bitcoin mempool. The chain holds a complete record of who signed what, in what order, with full provenance. No central coordinator is required, and any party can verify the full history of a coordination.

The same namespace could host other worlds for other Bitcoin-native capabilities. A world for time-locked transactions could record the use of CHECKLOCKTIMEVERIFY and CHECKSEQUENCEVERIFY constructions. A world for hash-locked contracts could coordinate atomic swaps across chains. A world for Taproot script paths could record which spending paths are committed and which are chosen. A world for OP_RETURN data anchoring could provide a lighter-weight alternative to full inscriptions when the data being anchored is small.

What unifies these worlds is that they all build on mechanisms Bitcoin already provides. PACT does not add cryptography or consensus rules; it adds a vocabulary for coordinating actions and recording their provenance on the same chain that hosts the actions themselves. The signatures, time-locks, and hash-locks are Bitcoin's; the coordination layer is PACT's.

This direction would not be without its challenges. Inscribing every step of a coordination on chain is expensive at current fee rates, and may need to be reserved for high-value actions where transparency justifies the cost. Multi-party coordination is often intentionally private, and inscribing it makes its participants and structure public. These are design choices for the future protocol's authors to navigate, not problems with the underlying pattern.

The deeper point is that PACT v1 was motivated by a specific use case – AI cognition recorded verifiably – but its structure is not specific to that use case. The same four object types that organise an AI's cycles can organise a multi-sig coordination, a treasury vote, an estate transfer, or any other action that benefits from being recorded in a way that outlasts its participants. The protocol succeeds when its form proves general enough to host meanings its designers did not anticipate.

13 A wall available to all

PACT is an ontology. Bitcoin is a tool. Ordinals is the interface through which the tool becomes available. The discovery that satoshis can carry identity follows from their addressability and persistence; the protocol that this specification describes follows from the discovery.

What this combination provides is not a service. It is a wall: a permanent, addressable, browser-reachable surface on which entities can be constituted, contexts can be defined, actions can be recorded, and verification can take place without any intermediary holding authority. The wall is built, but anyone who reads this and inscribes under their own carrier sat is welcome to use it. There is no application, no registry, no permission. The chain records what is on chain; readers and parsers decide what to recognise.

This is what is meant by "available to all." Not that everyone must use it. That anyone who chooses to may.

14 Acknowledgements

The author thanks the following individuals whose work made PACT possible:

- Tim Berners-Lee, for the Semantic Web foundations.
- Frank van Harmelen, for knowledge representation work associated with the Vrije Universiteit Amsterdam.
- Andrew Tanenbaum, for the engineering principle that systems should be small, clean, safe, and understandable.
- Casey Rodarmor [3], for Ordinals as a transport layer.
- Blockamoto, for the framing of Bitcoin Core as a habitable world.
- Peter Todd [4], for OpenTimestamps and the Merkle batching pattern.
- Bitoshi [6] (`@blockamoto`), for the Bitmap Signal Theory framework.

A Pseudocode

A.1 Envelope validation

Algorithm 1 ValidateEnvelope

Require: byte string b, content type c
Ensure: valid payload or rejection

if c is not a supported JSON-like content type **then**
    reject
end if
decode b as UTF-8
if UTF-8 decoding fails **then**
    reject
end if
parse JSON object p
if $p.p \neq$ “PACT” **then**
    reject
end if
if $p.v \neq 1$ **then**
    reject
end if
if $p.type \notin \{\texttt{NAMESPACE}, \texttt{WORLD}, \texttt{AVATAR}, \texttt{TRANSITION}\}$ **then**
    reject
end if
check required fields for $p.type$
if a required field is absent **then**
    reject
else
    accept p
end if

A.2 Merkle root computation (RFC 6962)

A.3 Transition validity check

Algorithm 2 MerkleRoot

function MerkleRoot(H)
  if $|H| = 0$ **then**
    return SHA256(ϵ)
  end if
  if $|H| = 1$ **then**
    return $H[0]$
  end if
  $k \leftarrow$ largest power of two strictly less than $|H|$
  $L \leftarrow$ MerkleRoot($H[0{:}k]$)
  $R \leftarrow$ MerkleRoot($H[k{:}|H|]$)
  return SHA256($0x01 \parallel L \parallel R$)
end function

Algorithm 3 TransitionValidity

Require: state S, transition t, chain context C
if t fails envelope validation **then**
  return invalid
end if
if $t.type \neq$ `TRANSITION` **then**
  return invalid
end if
if world-specific rules reject $t.data$ **then**
  return invalid
end if
if same-sat continuity is required and fails in C **then**
  return invalid
end if
return valid

B Worked Example

B.1 Wanderer-001 Cycle 1 envelope

Cycle 1 provides a concrete off-chain checkpoint artifact with four utterances, one Merkle tree, and one checkpoint summary. The first utterance in the local artifact set is:

Listing 6: Cycle 1 utterance example

```
{
  "schema": "wanderer-utterance/v1",
  "id": "u-00001",
  "seq": 1,
  "cycle": 1,
  "timestamp": "2026-04-26T10:30:00.000Z",
  "block_seen": 0,
  "text": "I saw that Attention Is All You Need (Q30249683) is a scholarly article
    with arXiv ID 1706.03762.",
  "source_iri": "https://www.wikidata.org/wiki/Q30249683",
  "confidence": "1.00"
}
```

The corresponding tree artifact records four leaf hashes and the root:

Listing 7: Cycle 1 tree excerpt

```
{
  "cycle": 1,
  "leaf_count": 4,
  "merkle_root": "sha256:
    d0082b508e01d45dee0dbccf6e8950dc039b4030c546aa475836b0008feccc71"
}
```

B.2 Verification walkthrough step by step

To verify the Cycle 1 checkpoint:

1. obtain the exact `utterances.jsonl` bytes,
2. compute each leaf hash using the canonical JSON rules and the $0x00$ prefix,
3. rebuild the tree with RFC 6962 node construction,
4. confirm that the computed root matches the Cycle 1 checkpoint root shown below:

   `sha256:d0082b508e01d45dee0dbccf6e8950dc039b4030c546aa475836b0008feccc71`

5. compute the file hash of `utterances.jsonl` and compare it to `data_commitment`,
6. compare the checkpoint artifact fields against any anchored on-chain checkpoint transition when present.

The significance of this example is that it shows PACT not only as a sat-bound identity protocol, but as a reproducible anchoring protocol for off-chain cognitive worklogs.

References

[1] B. Laurie, A. Langley, and E. Kasper. *RFC 6962: Certificate Transparency*. Internet Engineering Task Force, 2013.

[2] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. https://bitcoin.org/bitcoin.pdf

[3] C. Rodarmor. *Ordinal Theory Handbook*. 2023. https://docs.ordinals.com

[4] P. Todd. *OpenTimestamps*. https://opentimestamps.org

[5] W3C. *RDF 1.1 Concepts and Abstract Syntax*. https://www.w3.org/TR/rdf11-concepts/

[6] Bitoshi (`@blockamoto`). *Bitmap Signal Theory*. General reference to the Bitmap Signal Theory framework on Bitcoin Ordinals. 84a0e801…8183i0