**RFC-0001: Portable Tournament Data (PTD) Format**

| Category | Standards Track |
| --- | --- |
| **Status** | Draft |
| **Author** | Suparena Software Inc. |
| **Date** | 2025-09-25 |
| **Intended Audience** | Tournament software vendors, club systems, federation IT, researchers |

**Table of Contents**

1. Abstract
2. Terminology
3. PTD Package Structure
4. Envelope Format
5. Core Entities  
   5.1. Participants  
   5.2. Tournaments & Events  
   5.3. Rounds  
   5.4. Groups and Brackets  
   5.5. Matches  
   5.6. Score Events  
   5.7. Standings
6. Privacy Considerations
7. Internationalization
8. Integrity & Signatures
9. Canonicalization Rules
10. Database Mapping
11. Transport & Exchange
12. Extensibility
13. Security Considerations
14. Example Workflow
15. References
16. Reference Implementation (Go)

### 1. Abstract

This document specifies the Portable Tournament Data (PTD) format, a portable, verifiable, and interoperable standard for representing racket sport tournament data. PTD is designed for on-premise club systems, mobile/desktop clients, and cross-organization data exchange.

The format emphasizes the following key principles:

* **Portability**: Compact .ptd archives that can be exchanged offline or online.
* **Determinism**: Canonical serialization for reproducible hashes and signatures.
* **Extensibility**: Schema evolution without breaking compatibility with older data.
* **Privacy**: Separation of Personally Identifiable Information (PII) from competitive results.
* **Verifiability**: Manifest digests and optional digital signatures ensure data integrity.

### 2. Terminology

* **Entity**: A typed record (e.g., participant, match, round).
* **Envelope**: The metadata wrapper around each entity (ID, schema, version, refs).
* **NDJSON**: Newline-delimited JSON.
* **PTD Package**: A .ptd file (a ZIP archive with deterministic ordering).
* **ULID**: Universally Unique Lexicographically Sortable Identifier.
* **Append-only**: Data that can only be extended, never rewritten (e.g., score events).

### 3. PTD Package Structure

A PTD package is a ZIP archive with UTF-8 filenames, deterministic ordering, and no embedded timestamps to ensure reproducibility.

Plaintext

/manifest.json # package header, hashes, and sizes  
/meta/schema.json # JSON Schema bundle (optional)  
/org/participants.ndjson  
/org/clubs.ndjson  
/tournament/tournaments.ndjson  
/tournament/events.ndjson  
/tournament/rounds.ndjson  
/tournament/groups.ndjson  
/tournament/brackets.ndjson  
/tournament/matches.ndjson  
/tournament/score\_events.ndjson  
/tournament/standings.ndjson  
/resources/venues.ndjson  
/resources/tables.ndjson  
/resources/devices.ndjson  
/i18n/strings.ndjson  
/privacy/pii.ndjson  
/signature.sig # optional detached signature

### 4. Envelope Format

Every entity is wrapped in a common envelope structure, containing the entity body (spec) and associated metadata (meta) for validation, versioning, and linkage.

JSON

{  
 "id": "ptd:ulid:01J8H3Q2S3C3X1YV6Y45N1T9VZ",  
 "type": "match",  
 "spec": { /\* entity body \*/ },  
 "meta": {  
 "schema": "ptd.v1.match@1.0.0",  
 "version": 3,  
 "created\_at": "2025-08-24T15:04:05Z",  
 "updated\_at": "2025-08-24T16:11:21Z",  
 "source": "edge:club-oakville-01",  
 "etag": "W/\"a7f9b4c3\"",  
 "hash": "blake3:5b…",  
 "refs": { "tournament\_id": "ptd:…" }  
 }  
}

### 5. Core Entities

#### 5.1. Participants

Represents persons or teams competing in the tournament.

* **Kind**: person or team.
* **Attributes**: display\_name, nationality, club\_id, seed, rating.

#### 5.2. Tournaments & Events

* **Tournament**: The umbrella container for the entire competition (e.g., “2025 Oakville Open”).
* **Event**: A subdivision within the tournament (e.g., “Men's Singles” or “U13 Girls Singles”).

#### 5.3. Rounds

Defines the structure and progression of an Event (previously referred to as Stages).

* **Types**: group (round-robin), knockout or bracket (elimination), or qualifier.
* **Attributes**: Defines the format (e.g., best-of), tie-break policies, and participant lists for that phase.

#### 5.4. Groups and Brackets

Specific structures linked to Rounds.

* **Groups**: Define round-robin pools (stored in /tournament/groups.ndjson).
* **Brackets**: Define elimination structures or draws (stored in /tournament/brackets.ndjson).

#### 5.5. Matches

Represents a single contest between participants.

* **Attributes**: Participants, schedule, officials, current status.
* **Results**: The match entity may contain an optional snapshot result, but the score\_events (Section 5.6) are the authoritative source of truth for the match outcome.

#### 5.6. Score Events

An append-only, immutable log of events within a match.

* **Event Types**: Points, lets, timeouts, penalties, corrections.
* **Immutability**: Corrections are recorded as additional events, not edits to previous events.

#### 5.7. Standings

A computed projection of group or bracket outcomes. Standings must reference the specific tie-break algorithm identifier used for the calculation.

### 6. Privacy Considerations

* All Personally Identifiable Information (PII) is stored separately in /privacy/pii.ndjson.
* Exporters may omit this file to facilitate anonymized data exchange for statistical analysis or public display.
* Identifiers (ULIDs) and hashes allow linkage between competitive data and PII without exposing raw PII in the core data files.

### 7. Internationalization

* Translatable strings are stored centrally in /i18n/strings.ndjson.
* Entities may inline a display\_name for convenience but SHOULD reference i18n keys for localization support.

### 8. Integrity & Signatures

* Each NDJSON file within the package is hashed using a strong cryptographic hash (SHA-256 or BLAKE3).
* manifest.json lists all file hashes, file sizes, and the package version.
* An optional detached signature (/signature.sig) can be included to verify the authenticity and integrity of the entire package.

### 9. Canonicalization Rules

To ensure deterministic hashing and verification, the following rules must be applied during serialization:

* JSON keys must be sorted lexicographically.
* Arrays must preserve their defined order.
* Numbers must be normalized (e.g., no trailing zeros).
* Timestamps must be in RFC 3339 format and normalized to UTC (Z).
* NDJSON requires a newline character (\n) after each line, including the last line of the file.

### 10. Database Mapping

PTD entities map naturally to Relational Database Management System (RDBMS) tables:

* Tables: participants, tournaments, events, rounds, groups, brackets, matches, score\_events, standings.
* score\_events use a composite primary key of (match\_id, seq).
* Projections (e.g., standings, match results) are recomputed as jobs based on the core entities.

### 11. Transport & Exchange

PTD packages can be exchanged via:

* **HTTP**: GET /export/ptd, PUT /import/ptd.
* **Offline**: Removable media (e.g., USB drives).
* **Differential Updates**: Supported via NDJSON append operations and envelope version increments.

### 12. Extensibility

* Schema versions follow the format ptd.vX.entity@MAJOR.MINOR.
* New entity types MAY be introduced and defined in /meta/schema.json.
* Consumers MUST ignore unknown fields in known entity types but MUST preserve those fields during re-export to maintain forward compatibility.

### 13. Security Considerations

* Digital signatures (Section 8) prevent tampering and ensure authenticity.
* Optional encryption (e.g., ZIP AES or external transport-level encryption) can be used to protect PII during transit.
* The append-only event model prevents historical manipulation of match results.

### 14. Example Workflow

1. A club organizer runs a local tournament using a PTD-compliant backend (e.g., Go implementation storing data in PostgreSQL).
2. At the end of the tournament, the organizer exports the data as a .ptd archive.
3. The .ptd archive is shared via HTTP upload or offline media (USB stick) to the governing federation.
4. The federation server validates the package against the schema, verifies the signature, and imports the data into the master database.
5. A remote client or auditor can replay the score\_events locally to independently verify match integrity and outcomes.

### 15. References

* [NDJSON Specification](https://github.com/ndjson/ndjson-spec)
* [RFC 3339: Date and Time on the Internet](https://www.ietf.org/rfc/rfc3339.txt)
* [ULID Specification](https://github.com/ulid/spec)

### 16. Reference Implementation (Go)

A reference implementation is available in Go to assist developers in adopting PTD.

Location: github.com/suparena/ptd

The library provides:

* **Envelope & Entity Types**: Strongly typed Go structs (Envelope, Meta, MatchSpec, ScoreEventSpec, etc.).
* **Serialization**: JSON/NDJSON marshalling and unmarshalling with canonicalization rules applied.
* **Validation**: Schema validation against /meta/schema.json.
* **Storage Integration**: Helpers for mapping PTD entities to PostgreSQL or SQLite.
* **Utilities**: Functions to pack/unpack .ptd archives, compute manifest digests, and verify signatures.

#### Example Structs

Go

type Envelope[T any] struct {  
 ID string `json:"id"`  
 Type string `json:"type"`  
 Spec T `json:"spec"`  
 Meta Meta `json:"meta"`  
}  
  
type Meta struct {  
 Schema string `json:"schema"`  
 Version int `json:"version"`  
 CreatedAt time.Time `json:"created\_at"`  
 UpdatedAt time.Time `json:"updated\_at"`  
 Source string `json:"source,omitempty"`  
 ETag string `json:"etag,omitempty"`  
 Hash string `json:"hash,omitempty"`  
 Refs map[string]string `json:"refs,omitempty"`  
}  
  
type MatchSpec struct {  
 Participants []struct {  
 Role string `json:"role"`  
 ParticipantID string `json:"participant\_id"`  
 } `json:"participants"`  
 RoundID string `json:"round\_id"` // Standardized JSON key  
 BestOf int `json:"best\_of"`  
 GamesToWin int `json:"games\_to\_win"`  
 Status string `json:"status"`  
 Schedule struct {  
 StartAt time.Time `json:"start\_at"`  
 TableID string `json:"table\_id"`  
 } `json:"schedule"`  
 Officials struct {  
 RefereeID string `json:"referee\_id,omitempty"`  
 UmpireID string `json:"umpire\_id,omitempty"`  
 } `json:"officials"`  
 Result \*struct {  
 Sets []struct{ Home, Away int } `json:"sets"`  
 WinnerID string `json:"winner\_id"`  
 } `json:"result,omitempty"`  
}

Developers can import this package to quickly integrate PTD export, import, and validation workflows into their systems.

*End of RFC-0001*