Network Working Group
Internet-Draft
Intended status: Informational
Expires: 29 November 2026

Trimplayer Editors
Trimplayer

28 May 2026

PortCast: A JSON-Based Interchange Format and Sync API for Portable Podcast Listener Data

draft-trimplayer-portcast-00

Abstract

PortCast defines an open JSON-based interchange format, and an optional HTTPS synchronisation API, for moving a podcast listener's data -- subscriptions, listening history, playback position, queue, bookmarks, and per-feed preferences -- between independent podcast applications without a central service. It builds on identifiers already present in RSS (item GUID, feed URL) and the Podcast Namespace (podcast:guid) so that implementations can interoperate without inventing a new identity namespace. This document specifies the file format (v0.1) and a federated synchronisation API (v0.2) that reuses the same data model.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://portcast.org/. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-trimplayer-portcast/.

Source for this draft and an issue tracker can be found at https://github.com/Trim-Player/PortCast.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 29 November 2026.

Table of Contents

1. Introduction

A listener's relationship with their podcasts -- which shows they follow, where they stopped in an unfinished episode, the clip they bookmarked at 23:04 -- currently lives inside whichever application they happen to use. Switching applications restarts that relationship from zero. OPML [OPML2.0] solves the subscription case, but everything else (playback position, completion state, queue, bookmarks, per-feed preferences) is lost on every migration.

This document specifies PortCast, a protocol whose goal is simple: a listener SHOULD be able to leave any podcast application and arrive at any other application with the relationship to their podcasts intact. PortCast defines a JSON document format (file mode, Section 3 through Section 10) and an optional HTTPS API (API mode, Section 12) that exposes the same entities for incremental synchronisation. The two modes share a single data model; file mode is the interoperability floor that every conforming implementation can fall back to.

PortCast is intentionally federated. There is no central directory, registry, or authority. Each application exposes its own endpoint on its own domain, or produces its own files. The editors of this specification commit to not operating a central service.

1.1. Design goals

1. *Listener-owned.* The document is produced by the user, for the user. No vendor lock-in, no proprietary identifiers required.

2. *Interoperable identity.* Use open identifiers already present in RSS (item GUID, feed URL) rather than inventing a new namespace. Implementations MAY add their own identifiers in a namespaced extension block.

3. *Lossless within the model.* A conforming export captures everything the protocol defines. Anything outside the model goes in extensions so it round-trips through implementations that do not understand it.

4. *Partial and incremental.* Every entity carries an updatedAt timestamp, so a future synchronisation profile can ship deltas. The v0.1 file format is a full snapshot, but the data model is synchronisation-friendly.

5. *Human-readable.* A listener SHOULD be able to open the file in a text editor and recognise what it says about them.

6. *Versioned.* The document declares its protocol version; implementations can negotiate behaviour.

2. Terminology and conformance

2.1. Requirements language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Defined roles

A *producer* is software that writes a PortCast document or serves PortCast API responses.

A *consumer* is software that reads a PortCast document or calls a PortCast API.

A *conforming producer* MUST write a document that validates against the JSON Schemas published with this specification. A *conforming consumer* MUST accept any document that validates against those schemas, and MUST NOT reject a document because of unrecognised keys inside an extensions object (Section 10).

3. Document container

A PortCast document is a single JSON object [RFC8259], encoded in UTF-8 without a byte order mark, with the following top-level shape:

```
{
  "portcast": "0.1.0",
  "generatedAt": "2026-05-26T14:00:00Z",
  "generator": { "name": "Trimplayer", "version": "3.4.1" },
  "owner": { "displayName": "Jonathan", "email": "user@example.com" },
  "subscriptions": [ ],
  "episodes":      [ ],
  "queue":         [ ],
  "bookmarks":     [ ],
  "preferences":   { },
  "extensions":    { }
}
```

Field	Required	Notes
portcast	yes	Semantic-versioned string. The version of this spec.
generatedAt	yes	RFC 3339 [RFC3339] timestamp in UTC.
generator	yes	Producing application identifier.
owner	no	Optional listener identity. Producers SHOULD let users opt out.
subscriptions	yes	Array of Subscription. MAY be empty.
episodes	yes	Array of EpisodeState. MAY be empty.
queue	no	Ordered array of QueueItem.
bookmarks	no	Array of Bookmark.
preferences	no	Preferences object.
extensions	no	Namespaced extension data (Section 10).

Table 1

The file extension SHOULD be .portcast.json and the IANA media type SHOULD be application/vnd.portcast+json (see Section 14).

4. Identity model

Identity is the heart of an interoperability protocol. PortCast identifies entities at two levels.

4.1. Podcast identity

A Subscription MUST carry at least one of:

* feedUrl -- the canonical RSS or Atom URL of the show.

* podcastGuid -- the Podcast Namespace <podcast:guid> value [PODCAST-NAMESPACE] when the feed publishes one. This is the strongest identifier and SHOULD be preferred when matching across applications.

A subscription SHOULD include both when both are available. Producers MAY also carry directory-specific identifiers (e.g., Apple Podcasts, Podcast Index) under Subscription.identifiers.*; these are advisory and not required for matching.

4.2. Episode identity

An EpisodeState MUST carry at least one of:

* guid -- the RSS <item><guid> value (preferred).

* enclosureUrl -- the media URL from <enclosure url="...">.

It MUST also carry a subscriptionRef (either podcastGuid or feedUrl) that matches one of the document's subscriptions[] entries. If neither a guid nor an enclosureUrl is known, an episode state MAY use a stable (subscriptionRef, publishedAt, title) tuple, but consumers are not required to match by it.

5. Subscriptions

```
{
  "subscriptionId": "01HXYZ...",
  "feedUrl": "https://example.com/feed.xml",
  "podcastGuid": "917393e3-1b1e-5cef-ace4-edaa54e1f810",
  "title": "Example Podcast",
  "author": "Jane Doe",
  "imageUrl": "https://example.com/cover.jpg",
  "subscribedAt": "2024-06-01T09:14:00Z",
  "unsubscribedAt": null,
  "tags": ["tech", "weekly-listen"],
  "notificationsEnabled": true,
  "identifiers": {
    "applePodcastsId": "1500000000",
    "podcastIndexId": "920666"
  },
  "updatedAt": "2026-05-26T14:00:00Z"
}
```

unsubscribedAt is set when the listener has stopped following the show but the producer still wishes to convey historical context. Consumers MAY discard unsubscribed entries on import. tags are free-form, listener-applied labels.

6. Episode state

```
{
  "episodeStateId": "01HXYZ...",
  "subscriptionRef": { "podcastGuid": "917393e3-..." },
  "guid": "https://example.com/ep/42",
  "enclosureUrl": "https://example.com/audio/ep42.mp3",
  "title": "Episode 42",
  "publishedAt": "2026-05-20T07:00:00Z",
  "durationSeconds": 3287,

  "status": "in_progress",
  "positionSeconds": 1245.2,
  "playCount": 1,
  "completedAt": null,
  "firstPlayedAt": "2026-05-22T18:30:00Z",
  "lastPlayedAt": "2026-05-25T08:11:00Z",
  "rating": null,
  "starred": false,
  "hidden": false,

  "events": [
    { "type": "play",  "at": "2026-05-22T18:30:00Z", "positionSeconds": 0 },
    { "type": "pause", "at": "2026-05-22T19:05:12Z", "positionSeconds": 2112 }
  ],

  "updatedAt": "2026-05-25T08:11:00Z"
}
```

6.1. status values

Value	Meaning
unplayed	Listener has never started this episode.
in_progress	Listener started but did not finish.
completed	Listener reached the end or marked complete.
archived	Listener explicitly dismissed without listening.

Table 2

positionSeconds is REQUIRED when status is in_progress and SHOULD be zero or omitted otherwise. Producers SHOULD record a small tolerance (for example, treating the episode as completed if the listener reached within 30 seconds of the end).

6.2. Playback events

The events array is OPTIONAL. Each event has a type from the set {play, pause, seek, complete, speed_change, bookmark} and attaches its own typed fields. Producers MAY include a subset; consumers MAY ignore events they do not understand. Producers that do not track event-level history can omit events entirely; the top-level fields (positionSeconds, playCount, lastPlayedAt) are still sufficient for everyday "where did I leave off" portability.

7. Queue

```
{
  "queue": [
    { "position": 1, "episodeRef": { "guid": "https://example.com/ep/42" },
      "addedAt": "2026-05-25T09:00:00Z", "source": "manual" },
    { "position": 2, "episodeRef": { "enclosureUrl": "https://.../ep43.mp3" },
      "addedAt": "2026-05-25T09:01:00Z", "source": "auto" }
  ]
}
```

position is 1-based and MUST be unique within the queue. source is free-form (e.g., manual, auto, smart-playlist:Morning Commute).

8. Bookmarks

```
{
  "bookmarkId": "01HXYZ...",
  "episodeRef": { "guid": "https://example.com/ep/42" },
  "atSeconds": 1384.0,
  "endSeconds": 1421.5,
  "label": "Great quote about feed ownership",
  "note": "Quote starts at ’and if you can’t take it with you...’",
  "createdAt": "2026-05-25T08:23:00Z",
  "updatedAt": "2026-05-25T08:23:00Z"
}
```

endSeconds is OPTIONAL; its presence promotes a bookmark to a _clip_.

9. Preferences

```
{
  "preferences": {
    "global": {
      "playbackRate": 1.2,
      "skipForwardSeconds": 30,
      "skipBackwardSeconds": 15,
      "trimSilence": true,
      "boostVoice": false
    },
    "perFeed": {
      "917393e3-1b1e-5cef-ace4-edaa54e1f810": {
        "playbackRate": 1.0,
        "skipIntroSeconds": 90,
        "skipOutroSeconds": 60,
        "autoDownload": "latest-3"
      }
    }
  }
}
```

perFeed keys are podcastGuid when available, otherwise feedUrl. Per-feed values override global.

10. Extensions

Anything outside this specification lives under an extensions object, keyed by reverse-DNS namespace:

```
"extensions": {
  "com.trimplayer.skips": [
    { "episodeGuid": "...", "skippedRanges": [[12.0, 47.5]] }
  ],
  "fm.overcast.smart-speed": { "secondsSaved": 18421 }
}
```

Consumers MUST preserve extensions on round-trip even if they do not understand a namespace. This is what keeps a multi-application journey lossless.

11. Versioning

portcast is a semantic-versioned string. Consumers:

* MUST accept any document whose portcast major version they support.

* MAY warn the listener when a minor version is newer than they understand.

* MUST NOT silently drop fields they do not recognise; preserve them under extensions._unknown if necessary.

12. Live sync API (v0.2 -- Draft)

The file format defined above (v0.1) is the interoperability floor. v0.2 introduces an optional API mode: the same entities, exposed over HTTPS so clients can synchronise incrementally without a full re-export. The wire payloads in API mode reuse the v0.1 schemas; no new entity shapes are introduced.

A conforming v0.2 implementation MAY implement file mode, API mode, or both. Clients MUST assume nothing beyond what a server advertises in its discovery document (Section 12.2).

12.1. Operating modes

Mode	Transport	Use case
File	User-supplied .portcast.json	One-shot migration, archival, manual transfer
API	HTTPS endpoints on the application’s domain	Live sync between two installed applications

Table 3

File mode is the interoperability floor: every API-mode server SHOULD implement at least GET /portcast/v1/export, which returns the same document a file export would produce.

12.2. Discovery

A PortCast server SHOULD publish a discovery document at /.well-known/portcast [RFC8615]:

```
{
  "portcast": "0.2.0",
  "base": "https://example.app/portcast/v1",
  "auth": {
    "type": "oauth2",
    "authorizationEndpoint": "https://example.app/oauth/authorize",
    "tokenEndpoint": "https://example.app/oauth/token",
    "scopes": ["portcast.read", "portcast.write", "portcast.history"]
  },
  "capabilities": [
    "export",
    "subscriptions.read", "subscriptions.write",
    "episodes.read", "episodes.write",
    "queue.read", "queue.write",
    "bookmarks.read", "bookmarks.write",
    "preferences.read", "preferences.write",
    "events", "deltas", "webhooks"
  ]
}
```

capabilities is a flat string set. A read-only server omits *.write entries. A server that does not track per-event history omits events. A server that does not implement delta synchronisation omits deltas and clients fall back to fetching full collections.

12.3. Versioning and content type

Endpoints live under /portcast/v1/.... The v1 is the API major version and is independent of the specification version declared inside payloads. Servers MUST set Content-Type: application/vnd.portcast+json on responses; clients SHOULD send a matching Accept header. Backwards-compatible additions (new optional fields, new capability strings) MUST NOT bump the API major version.

12.4. Authentication

Implementations MUST use one of:

* OAuth 2.0 [RFC6749], with scopes drawn from portcast.read, portcast.write, portcast.history. The portcast.history scope covers event-level playback data (Section 6.2) and is treated as more sensitive than basic read.

* Bearer token [RFC6750], appropriate for self-hosted or single-user deployments.

Credentials MUST NOT appear in URL query strings. Servers MUST reject requests over plain HTTP.

12.5. Endpoints

Method	Path	Body / Returns
GET	/portcast/v1/export	Full PortCast document (= v0.1 file)
POST	/portcast/v1/import	Body: full or partial PortCast document; server upserts each entity
GET	/portcast/v1/subscriptions	{ subscriptions, deletions?, syncedAt, nextCursor? }
GET	/portcast/v1/subscriptions/{ref}	A Subscription
PUT	/portcast/v1/subscriptions/{ref}	Body: Subscription
DELETE	/portcast/v1/subscriptions/{ref}	Unsubscribe
GET	/portcast/v1/episodes	{ episodes, deletions?, syncedAt, nextCursor? }
POST	/portcast/v1/episodes	Body: { episodes: [...] }; upsert
GET	/portcast/v1/queue	{ queue }
PUT	/portcast/v1/queue	Body: { queue }; replaces in full
GET	/portcast/v1/bookmarks	{ bookmarks, deletions?, syncedAt, nextCursor? }
POST	/portcast/v1/bookmarks	Body: Bookmark
DELETE	/portcast/v1/bookmarks/{bookmarkId}	
GET	/portcast/v1/preferences	Preferences
PUT	/portcast/v1/	Body: Preferences

	preferences	

Table 4

{ref} in subscription paths is the URL-encoded podcastGuid when known, otherwise the URL-encoded feedUrl. Servers MUST accept either form. Episodes are intentionally not addressed by path because RSS GUIDs do not round-trip cleanly through URL encoding.

12.6. Delta sync

Collection endpoints (subscriptions, episodes, bookmarks) MUST accept ?since=<RFC 3339 timestamp> when the server advertises the deltas capability. The response then contains only entities whose updatedAt > since, a deletions array of refs for entities removed since that timestamp, and a syncedAt timestamp the client persists for the next round.

Servers SHOULD retain deletion tombstones for at least 30 days. Clients that have been offline longer SHOULD discard their cached syncedAt and perform a full pull.

12.7. Pagination

Endpoints that may return large collections support cursor-based pagination. The response carries nextCursor when more pages exist; the client passes ?cursor=<value> to fetch the next page. Cursors are opaque strings. since and cursor MAY be combined.

12.8. Conditional updates

Writes SHOULD use If-Match: <updatedAt> for optimistic concurrency. Servers MUST respond 412 Precondition Failed if the resource’s current updatedAt is newer than the supplied value. This prevents two clients clobbering each other’s position updates on the same episode.

12.9. Errors

Errors are JSON, with HTTP status reflecting the class:

```
{
  "error": {
    "code": "subscription_not_found",
    "message": "No subscription matched podcastGuid=917393e3-...",
    "ref": { "podcastGuid": "917393e3-..." }
  }
}
```

The initial set of code values is registered in Section 14.4. Servers MAY define additional codes under a reverse-DNS prefix (e.g., com.example.quota_exceeded); such vendor-prefixed codes do not require IANA registration.

12.10. Webhooks (optional)

Servers advertising the webhooks capability accept registrations at POST /portcast/v1/webhooks with body:

```
{
  "url": "https://client.example/portcast/hook",
  "events": ["episode.updated", "subscription.added",
             "subscription.removed", "queue.updated"],
  "secret": "<shared secret, >= 32 bytes>"
}
```

Webhook deliveries carry X-PortCast-Signature: sha256=<hex> computed over the raw request body with the registration secret as the HMAC key. Receivers MUST verify the signature and SHOULD respond 2xx within 5 seconds. Servers SHOULD retry failed deliveries with exponential backoff for at least 24 hours.

Webhooks are an optimisation; the baseline pattern is client-driven polling with ?since=.

12.11. Capability fallback

A client that needs a capability the server does not advertise SHOULD fall back to GET /portcast/v1/export and process the returned document as a file-mode import. This guarantees a baseline interoperability floor even for minimal server implementations.

12.12. Federation

PortCast is intentionally federated. Each application exposes its own endpoint on its own domain; there is no central directory or hub. A client connecting a new account typically:

1. Asks the listener for the application’s domain.

2. Fetches https://<domain>/.well-known/portcast.

3. Runs the OAuth dance against the endpoints declared there.

4. Begins delta-synchronising.

Servers MUST NOT require registration with any central authority to be considered conforming.

13. Security considerations

13.1. Sensitivity of the data

A PortCast document is a detailed record of personal listening behaviour: which shows a listener follows, when they started or finished an episode, which passages they bookmarked, and (when event-level history is included) the moment-to-moment shape of their attention. Implementers MUST treat PortCast documents and API responses as personal data of comparable sensitivity to browser history or messaging metadata.

13.2. Producer requirements

Producers:

* SHOULD let the listener choose whether to include owner.

* SHOULD let the listener choose whether to include event-level history (Section 6.2).

* SHOULD NOT include device identifiers, IP addresses, geolocation, or third-party analytics identifiers in any PortCast field, including inside extensions.

* MUST NOT embed the listener’s account credentials, API keys, OAuth tokens, or session cookies anywhere in a PortCast document.

* SHOULD warn the listener before transmitting a PortCast document to a third party.

13.3. Consumer requirements

Consumers SHOULD treat an imported document as personal data, not as shareable telemetry. Consumers MUST NOT retransmit a received document to third parties without explicit listener consent. Consumers SHOULD make it possible for the listener to delete imported data on demand.

13.4. Transport and storage (API mode)

In API mode (Section 12):

* Servers MUST require TLS; plain HTTP MUST be refused.
* Credentials MUST NOT appear in URL query strings or path components; they MUST be carried in HTTP request headers.
* Servers MUST scope OAuth tokens to a single listener account.
* Servers SHOULD support per-client token revocation.
* The portcast.history scope SHOULD be requested separately from portcast.read.

13.5. Threat model and out-of-scope risks

PortCast does not, in v0.1, define a signing or sealing mechanism: a document cannot be cryptographically attributed to the producer that wrote it. Consumers SHOULD treat the source of a document as out-of-band-authenticated (e.g., the listener manually selected the file or authorised the OAuth client). Adding a signed manifest is listed as a future work item.

PortCast does not protect against a malicious application that has been granted access to a listener’s data; access control is the responsibility of the producing or hosting application, not the protocol. Consumers SHOULD apply input validation to imported documents (notably to URL fields and extensions content) consistent with their platform’s safe-handling guidance.

14. IANA considerations

This document requests four IANA actions.

14.1. Media type registration

IANA is requested to register the following media type per [RFC6838]:

Field	Value
Type name	application
Subtype name	vnd.portcast+json
Required parameters	none
Optional parameters	none
Encoding considerations	binary; PortCast documents are UTF-8 encoded JSON
Security considerations	See Section 13 of this document
Interoperability cons.	See Section 11 of this document
Published specification	This document
Applications that use it	Podcast applications, subscription importers and exporters, listener-data synchronisation services
Fragment identifier	JSON Pointer syntax [RFC6901]
Restrictions on use	none
Provisional registration	yes (until RFC publication)
Author / change controller	The editors of this specification
Intended usage	COMMON

Table 5

The file extension .portcast.json is the RECOMMENDED extension; the +json structured-syntax suffix indicates the underlying JSON serialization.

14.2. Well-known URI registration

IANA is requested to register a new entry in the "Well-Known URIs" registry per [RFC8615]:

Field	Value
URI suffix	portcast
Change controller	The editors of this specification
Specification document	This document (Section 12.2)
Related information	The resource is a JSON object describing a PortCast API endpoint (its base URL, authentication scheme, and capability set)
Status	provisional

Table 6

14.3. OAuth 2.0 scope registration

IANA is requested to register the following OAuth 2.0 scopes per [RFC6749] and [RFC8809]:

Scope name	Description
portcast.read	Read subscriptions, episode state (excluding event history), queue, bookmarks, and preferences
portcast.write	Create, update, and delete the same entities the portcast.read scope grants visibility into
portcast.history	Read or write event-level playback history (Section 6.2). MUST be requested separately from portcast.read

Table 7

Change controller: the editors of this specification.

14.4. PortCast error code registry

This document establishes a new IANA registry titled "PortCast Error Codes" with the following structure:

Field	Type / notes
code	A short, lowercase, underscore-separated identifier returned in API error responses (Section 12.9)
description	A one-sentence summary of when the error is returned
reference	The document defining the code

Table 8

The registration policy is Specification Required [RFC8126]. Initial contents:

Code	Description	Reference
unauthorized	The request lacks valid authentication credentials	This document
forbidden	The credentials do not grant access to the resource	This document
not_found	The referenced entity does not exist	This document
conflict	The request conflicts with current server state	This document
precondition_failed	An If-Match precondition was not satisfied (Section 12.8)	This document
invalid_request	The request body or parameters are malformed	This document
unsupported_capability	The client asked for a capability the server does not advertise	This document
rate_limited	The client has exceeded a server-defined rate limit	This document
internal_error	The server encountered an unexpected error	This document

Table 9

15. References

15.1. Normative References

[JSON-SCHEMA-2020-12]
Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents (Draft 2020-12)", n.d., <https://json-schema.org/draft/2020-12/schema>.

[PODCAST-NAMESPACE]
Podcasting 2.0 Project, "The ’podcast’ Namespace - podcast:guid element", n.d., <https://podcastindex.org/namespace/1.0#guid>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/rfc/rfc3339>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/rfc/rfc6901>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC8809] Hodges, J., Mandyam, G., and M. Jones, "Registries for Web Authentication (WebAuthn)", RFC 8809, DOI 10.17487/RFC8809, August 2020, <https://www.rfc-editor.org/rfc/rfc8809>.

15.2. Informative References

[OPML2.0] Winer, D., "OPML 2.0 Specification", October 2007, <http://opml.org/spec2.opml>.

[RFC4846] Klensin, J., Ed. and D. Thaler, Ed., "Independent Submissions to the RFC Editor", RFC 4846, DOI 10.17487/RFC4846, July 2007, <https://www.rfc-editor.org/rfc/rfc4846>.

[RFC5378] Bradner, S., Ed. and J. Contreras, Ed., "Rights Contributors Provide to the IETF Trust", BCP 78, RFC 5378, DOI 10.17487/RFC5378, November 2008, <https://www.rfc-editor.org/rfc/rfc5378>.

[RFC5744] Braden, R. and J. Halpern, "Procedures for Rights Handling in the RFC Independent Submission Stream", RFC 5744, DOI 10.17487/RFC5744, December 2009, <https://www.rfc-editor.org/rfc/rfc5744>.

[RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, DOI 10.17487/RFC7033, September 2013, <https://www.rfc-editor.org/rfc/rfc7033>.

Acknowledgments

PortCast builds on a long tradition of attempts to make a listener’s relationship with their podcasts portable. The editors thank Dave Winer for OPML, which has carried podcast subscriptions across applications for two decades and which inspired the goal of doing the same for the rest of a listener’s data. The editors also thank the Podcast Namespace project for <podcast:guid>, which makes cross-application show identity tractable, and the podcast-application development community for feedback on early drafts.

Author’s Address

Trimplayer Editors
Trimplayer

Email: trimplayerapp@gmail.com
URI: https://trimplayer.com/