httpapi
Internet-Draft
Intended status: Standards Track
Expires: 30 August 2026

G. Ratnawat
IMTF

26 February 2026

Problem Details for Asynchronous Job Failures
draft-ratnawat-httpapi-async-problem-details-00

Abstract

HTTP APIs that process work asynchronously need a standard way to report job failures. "Problem Details for HTTP APIs" (RFC 9457) provides the envelope; this document defines extension members that fill it with asynchronous-job-specific context.

Eight extension members are specified: "jobId", "jobStatus", "submittedAt", "completedAt", "retryable", "retryAfter", "processingStage", and "correlationId". A ninth member, "results", supports batch operations. Together they let a server describe which job failed, when, at what pipeline stage, and whether a retry is advisable -- in a single, machine-readable JSON (RFC 8259) object that works equally well in an HTTP response body, a message-broker payload, or a webhook callback.

Although the primary motivation is structured error reporting for failed jobs, the extension members are equally useful for communicating successful job outcomes (e.g., a COMPLETED status with timing information).

This document does NOT redefine how to submit, poll, or cancel asynchronous jobs; those mechanics are already covered by "HTTP Semantics" (RFC 9110) (202 Accepted), "Prefer Header for HTTP" (RFC 7240) (respond-async), and emerging IETF work on long-running operations. Instead, it focuses exclusively on the structured reporting gap that remains after a job reaches a terminal state.

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/.

Copyright Notice

Table of Contents

1. Introduction

Asynchronous job processing is a pervasive pattern in HTTP APIs. When a server cannot fulfil a request within a single request-response cycle, it accepts the work, processes it in the background, and reports the outcome later.

The IETF has standardized the mechanics of this pattern well:

* [RFC9110] Section 15.3.3 defines the "202 Accepted" status code for requests accepted for processing.

* [RFC7240] Section 4.1 defines the "respond-async" preference token that lets clients opt into asynchronous handling.

* [RFC8288] defines link relations that can point to status polling resources.

What has NOT been standardized is the structure of error reports when asynchronous jobs fail.

[RFC9457] provides a general-purpose envelope for HTTP API errors ("Problem Details"), including the ability to define extension members for domain-specific context. However, no specification defines reusable extension members for the asynchronous job domain -- leaving every API to invent its own.

This document defines exactly those extension members.

The design principle is transport independence: the same Problem Details object -- with the same extension members -- can appear in an HTTP response, a Kafka message, a webhook POST body, or a Server-Sent Event. This is critical because asynchronous job results often travel through non-HTTP transports where HTTP headers like "Retry-After" are unavailable.

1.1. What This Document Is NOT

To avoid duplicating existing and emerging IETF work, this document explicitly does NOT define:

* How to submit an asynchronous job. Use POST with "Prefer: respond-async" [RFC7240] and "202 Accepted" [RFC9110].

* How to poll for job status. Use the URI from the "Location" header of the 202 response, or a link relation pointing to the job status resource [RFC8288].

* How to cancel a running job. This is operation-specific.

* Idempotency semantics for job submission. See [I-D.ietf-httpapi-idempotency-key-header].

* Rate-limiting headers for status polling. See [I-D.ietf-httpapi-ratelimit-headers] and [RFC6585] (429).

* A new media type. This document uses "application/problem+json" as defined by [RFC9457].

This document fills a single, specific gap: structured error context for async job failures, expressed as [RFC9457] extension members.

1.2. Relationship to Existing Standards

The following table maps each aspect of the asynchronous job lifecycle to the standard that covers it, and identifies where this document contributes.

Lifecycle Aspect	Covered By	This Document
Accepting a job	RFC 9110 (202)	No (uses as-is)
Client async preference	RFC 7240	No (uses as-is)
Status polling link	RFC 8288	No (uses as-is)
Idempotent submission	I-D.ietf-httpapi-idempotency-key	No (uses as-is)
Rate-limited polling	I-D.ietf-httpapi-ratelimit-headers	No (uses as-is)
Error envelope format	RFC 9457	No (extends)
Job failure context	(none)	YES — defines extension members
Transport-independent retry semantics	(none)	YES — defines retryable, retryAfter
Batch partial failure structure	(none)	YES — defines results array

Table 1: Relationship to Existing Standards

1.3. Notational Conventions

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.

This document uses the terms "extension member", "problem type", and "problem details object" as defined in [RFC9457].

JSON [RFC8259] is used for all examples and the normative schema. Whitespace in JSON examples is insignificant and is included for readability only.

1.4. Terminology

job: A unit of work that a server accepts for asynchronous processing in response to a client request.

job status resource: An HTTP resource whose representation describes the current state of a job. Typically the target of the "Location" header returned in a "202 Accepted" response.

terminal state: A job status from which no further transitions are possible (COMPLETED, FAILED, CANCELLED, TIMED_OUT, COMPLETED_WITH_ERRORS).

non-terminal state: A job status from which the job may still transition (ACCEPTED, PROCESSING, and any server-defined intermediate states).

originating request: The HTTP request that caused the job to be created.

2. Problem Statement

2.1. The Gap in RFC 9457

[RFC9457] defines five standard members for problem details objects: "type", "title", "status", "detail", and "instance". It explicitly allows extension members for domain-specific data (Section 3.2 of [RFC9457]).

However, [RFC9457] does not define any reusable extension members. Each API defines its own, leading to fragmentation. For the asynchronous job domain -- one of the most common API patterns -- this fragmentation is particularly costly because:

* Monitoring tools cannot parse job failure context across APIs.

* Client libraries must handle bespoke error shapes per service.

* API designers spend time reinventing the same fields.

2.2. The Transport-Independence Problem

A key challenge specific to asynchronous jobs is that the failure report may not travel over HTTP at all. Consider:

* A PDF generation service publishes job results to a Kafka topic. There are no HTTP headers; the entire context must be in the message payload.

* A webhook delivers the result via a POST callback. The "Retry-After" header applies to the webhook delivery, not to the original job.

* A Server-Sent Event stream pushes status updates. The event data field carries JSON; HTTP headers are from the SSE connection, not the job.

The "Retry-After" header field ([RFC9110] Section 10.2.3) is transport-bound: it only exists in HTTP responses. When a job failure must be communicated via Kafka, SSE, gRPC, or a webhook, the retry signal is lost.

This document solves this by defining "retryable" and "retryAfter" as JSON members inside the Problem Details object -- making them available regardless of transport.

2.3. Survey of Current Practice

The following table surveys how four major platforms represent async job failures today. No two use the same field names.

Concept	AWS Step Functions	Azure LRO	Google AIP-151	Stripe
Job identifier	executionArn	id	name	id
Status field	status	status	done + error	status
Status values	RUNNING, SUCCEEDED, FAILED, TIMED_OUT, ABORTED	Succeeded, Failed, Cancelled	done: true/false	pending, succeeded, failed
Submission time	startDate	createdDateTime	metadata.createTime	created
Completion time	stopDate	lastUpdatedDateTime	metadata.endTime	N/A
Error structure	error, cause (strings)	error: {code, message}	google.rpc.Status	error: {type, message}
Retry guidance	N/A	retryAfter	N/A	N/A
Failure stage	N/A	N/A	N/A	N/A
Error format	Custom JSON	Custom JSON	google.rpc.Status	Custom JSON

Table 2: Async Job Error Reporting — Industry Survey

Key observations:

* Every platform invents its own field names, status values, and error structures.

* None builds on [RFC9457], despite it being the IETF standard for HTTP API errors.

* Retry guidance is almost never structured (only Azure’s LRO pattern includes "retryAfter", and only in the HTTP header).

* Processing stage information -- which pipeline step failed -- is universally absent, despite being essential for debugging multi-step workflows (e.g., validate → render → convert → store).

* Google's AIP-151 (Long-Running Operations) comes closest to a standard but is Google-specific and uses gRPC/Protobuf, not [RFC9457].

3. Extension Members

3.1. Overview

This section defines eight extension members for [RFC9457] problem details objects. They are organized into four groups:

Identification: "jobId" (Section 3.2) and "correlationId" (Section 3.9).

State and Timing: "jobStatus" (Section 3.3), "submittedAt" (Section 3.4), and "completedAt" (Section 3.5).

Retry Guidance: "retryable" (Section 3.6) and "retryAfter" (Section 3.7).

Diagnostics: "processingStage" (Section 3.8).

All extension members are OPTIONAL in any given problem details object. A server MAY include any subset of them. A client MUST NOT assume any particular member is present and MUST gracefully handle its absence.

When present, these members MUST appear at the top level of the problem details JSON object, alongside the standard [RFC9457] members ("type", "title", "status", "detail", "instance").

Although defined as [RFC9457] extension members, these JSON member names and semantics MAY also be used in non-problem-details JSON objects (e.g., successful job status responses). When used outside a problem details context, the [RFC9457] standard members ("type", "title", "status", "detail", "instance") are not required. However, when reporting failures, "application/problem+json" SHOULD be used per [RFC9457].

In all non-HTTP transport contexts (message brokers, webhooks, SSE), the "status" member represents the HTTP status code that WOULD have been used if the failure had been reported in a synchronous HTTP response. It is not the HTTP status code of the delivery mechanism.

3.1.1. Conformance Levels

To provide interoperability guidance while preserving flexibility, this document defines two conformance levels:

Basic Conformance: A problem details object that includes at least one extension member from this specification. This is the minimum bar: any server that includes "jobId", "processingStage", or any other member defined herein is using this specification.

Full Conformance: A problem details object that includes "jobId", "jobStatus", and "submittedAt". These three members together provide sufficient context to uniquely identify a job, know its state, and establish when it was accepted. Servers that aim for interoperability across monitoring tools and client libraries SHOULD target Full Conformance.

Individual member definitions below use "RECOMMENDED" to indicate members that SHOULD be present for Full Conformance. This is not a contradiction with the OPTIONAL preamble above: all members remain OPTIONAL in the [RFC9457] sense, but Full Conformance requires a specific subset.

3.2. "jobId"

A string that uniquely identifies the asynchronous job.

* Type: string

* Constraints: The server MUST assign this value when the job is accepted. The value SHOULD be a UUID [RFC9562] or another globally unique identifier. The client MUST treat it as an opaque string and MUST NOT parse or interpret its structure.

* When to include: RECOMMENDED whenever reporting an async job outcome.

* Relationship: If the job was submitted via HTTP and the server returned a "Location" header pointing to a job status resource, the "jobId" value SHOULD match the identifier embedded in that URI.

Example:

"jobId": "550e8400-e29b-41d4-a716-446655440000"

3.3. "jobStatus"

A string indicating the current state of the job. Registered values are defined in Section 4.

* Type: string (case-sensitive, enumerated)

* Constraints: Consumers MUST perform case-sensitive string comparison when matching "jobStatus" values against the registered values in Section 4.

* When to include: RECOMMENDED whenever reporting an async job outcome.

Example:

"jobStatus": "FAILED"

3.4. "submittedAt"

An [RFC3339] timestamp indicating when the server accepted the job for processing.

* Type: string (date-time per [RFC3339])

* Constraints: MUST be in UTC (indicated by the "Z" suffix) to avoid timezone ambiguity in distributed systems.

* When to include: RECOMMENDED whenever reporting an async job outcome.

Example:

"submittedAt": "2026-02-26T10:00:00Z"

3.5. "completedAt"

An [RFC3339] timestamp indicating when the job reached a terminal state.

* Type: string (date-time per [RFC3339])

* Constraints: MUST be in UTC. MUST NOT be present when the job is in a non-terminal state. When the problem details object is published to a message broker or delivered via webhook, the producer MUST ensure that "completedAt" is absent if "jobStatus" is a non-terminal value at the time of publication. Consumers that encounter this inconsistency SHOULD log a warning and ignore the "completedAt" value.

* When to include: RECOMMENDED when the job has reached a terminal state.

Example:

"completedAt": "2026-02-26T10:00:05Z"

3.6. "retryable"

A boolean indicating whether the client SHOULD retry the job submission with the same input.

* Type: boolean

* Default: If absent, the client SHOULD assume "false" (not retryable).

* Semantics:

  - "true": The failure is likely transient (e.g., a downstream dependency was temporarily unavailable). The same request MAY succeed if resubmitted.

  - "false": The failure is deterministic (e.g., invalid input, missing template). Retrying with the same input will produce the same failure.

* Relationship to "Retry-After" header: When the failure is reported in an HTTP response, the server SHOULD also include the "Retry-After" header field [RFC9110] with a value consistent with the "retryAfter" extension member. When the failure is reported over a non-HTTP transport (message broker, webhook, SSE), the "retryAfter" member is the only mechanism available. This is the primary motivation for expressing retry semantics as a JSON member rather than relying solely on the HTTP header.

Example:

"retryable": true

3.7. "retryAfter"

A non-negative integer indicating the number of seconds the client SHOULD wait before retrying the job submission.

* Type: integer (>= 0)

* Constraints:

  - SHOULD NOT be present when "retryable" is absent or "false". In exceptional cases, a server MAY include "retryAfter" without "retryable" to signal a general backoff interval (e.g., the server is under load and wants all clients to slow down), but clients SHOULD NOT interpret this as permission to retry a deterministically failed job.

  - SHOULD be present when "retryable" is "true".

  - A value of 0 is syntactically valid but clients MUST NOT interpret it as "retry immediately"; see Section 9.4 for minimum backoff requirements.

* Semantics: Equivalent to the "Retry-After" header field ([RFC9110] Section 10.2.3) expressed as seconds, but embedded in the JSON body for transport independence.

* Temporal reference: The "retryAfter" value represents the minimum number of seconds to wait before retrying, measured from the value of "completedAt" (if present). When "completedAt" is absent, the value is measured from the time the consumer receives the problem details object. For non-HTTP transports where delivery delay is possible (e.g., a lagging Kafka consumer), consumers SHOULD compute the effective wait time as: retryAfter - (now - completedAt). If the result is zero or negative, the consumer MAY retry immediately (subject to the minimum backoff floor in Section 9.4).

* Client-side safety: Regardless of the value of "retryAfter", clients MUST enforce a minimum backoff floor (e.g., 1 second) to prevent tight retry loops in case of a misconfigured or malicious server. Clients SHOULD also enforce a maximum retry delay (e.g., 3600 seconds) beyond which the "retryAfter" value is treated as indicating that a retry is impractical. See Section 9.4.

* Relationship to draft-ietf-httpapi-ratelimit-headers: The RateLimit header fields defined in that draft apply to the HTTP request rate for the polling endpoint. The "retryAfter" extension member defined here applies to the job resubmission interval. These are distinct concerns: one governs how often you ask "is it done?", the other governs how soon you should try the work again.

Example:

"retryAfter": 30

3.8. "processingStage"

A string identifying the stage in the server’s processing pipeline at which the job failed or completed.

* Type: string

* Constraints: The value is server-defined. Servers that use this member SHOULD document the possible values and their meanings in their API specification.

* When to include: OPTIONAL. RECOMMENDED when the server has a multi-step processing pipeline and the failure can be attributed to a specific step.

* Registry: This document does not create an IANA registry for processing stage values. Stage names are inherently API-specific because processing pipelines vary too widely across domains. The interoperability benefit of this member comes from its structured presence (enabling tools to extract and display the stage), not from standardized values. Servers SHOULD use the stage names from Table 3 when applicable, and MAY define additional names for domain-specific stages.

Recommended stage names (servers MAY define others):

Stage	Description
validation	Input validation against schema or business rules failed.
authorization	The job’s security context was insufficient for the requested operation.
queuing	The job could not be placed onto the processing queue (broker unavailable, queue full).
processing	General processing failure (use more specific stages when possible).
rendering	Template or document rendering failed.
conversion	Format conversion failed (e.g., HTML to PDF, image transcoding).
storage	The result could not be persisted (disk, object store, database).
delivery	The result could not be delivered to the callback URL or output channel.

Table 3: Recommended Processing Stage Names

Example:

"processingStage": "rendering"

3.9. "correlationId"

A string containing a client-supplied identifier from the originating request, used for distributed tracing.

* Type: string

* Constraints: The value is client-supplied and MUST be treated as untrusted (see Section 9.2). Servers SHOULD enforce a maximum length (e.g., 256 characters) and reject or truncate values that exceed it. See Section 9.2.

* When to include: RECOMMENDED when the originating request included a correlation identifier (e.g., via the "X-Correlation-ID" header, the W3C "traceparent" header [W3C.TRACE-CONTEXT], or a field in the request body).

* Relationship to W3C Trace Context: If the server supports W3C Trace Context [W3C.TRACE-CONTEXT], the "correlationId" SHOULD carry the trace-id portion of the "traceparent" header. This allows the problem details object to be correlated with distributed traces without requiring the consumer to have access to the originating HTTP headers.

Example:

"correlationId": "4bf92f3577b34da6a3ce929d0e0e4736"

4. Job Status Registry

4.1. Initial Values

Status Value	Description
ACCEPTED	The job has been accepted but processing has not yet started.
PROCESSING	The job is actively being processed.
COMPLETED	The job finished successfully; the result is available.
FAILED	The job encountered an error. The "detail" member SHOULD describe the failure.
CANCELLED	The job was cancelled before completion.
TIMED_OUT	The job exceeded the server’s maximum allowed processing duration.
COMPLETED_WITH_ERRORS	(Batch only) Some items succeeded and some failed. See Section 7.

Table 4: Initial Job Status Values

4.2. Terminal vs. Non-Terminal States

Terminal states: COMPLETED, FAILED, CANCELLED, TIMED_OUT, COMPLETED_WITH_ERRORS. Once a job reaches a terminal state, its "jobStatus" MUST NOT change.

Non-terminal states: ACCEPTED, PROCESSING.

4.3. Extensibility

Servers MAY define additional status values for their domain (e.g., "VALIDATING", "RENDERING", "AWAITING_APPROVAL").

To reduce the risk of interoperability issues from typos or inconsistent naming, server-defined status values SHOULD use UPPER_SNAKE_CASE to match the registered values, and SHOULD be documented in the API specification.

Clients that encounter an unrecognized status value SHOULD treat it as non-terminal (equivalent to "PROCESSING") unless the value is documented by the API as terminal.

5. Transport Contexts

A core design goal of this specification is that the extension members work identically regardless of how the problem details object reaches the consumer. This section provides guidance for four common transports.

5.1. HTTP Response Bodies

This is the primary context anticipated by [RFC9457]. The problem details object appears in the body of an HTTP response with Content-Type "application/problem+json".

When used in HTTP responses, servers SHOULD also set:

* The "Retry-After" header field to match the "retryAfter" member (if present), for compatibility with clients that do not parse the body.

* The "status" member to match the HTTP status code of the response, per [RFC9457] Section 3.1.

Example context: A client polls a job status resource and receives a failure report.

5.2. Message Broker Payloads

When a job result is published to a message broker (e.g., Apache Kafka, RabbitMQ, Amazon SQS), the problem details object is serialized as the message value.

In this context:

* HTTP headers are unavailable. The "retryAfter" member is the only mechanism for conveying retry guidance.

* The "status" member represents the HTTP status code that WOULD have been used if the failure had been reported synchronously.

* The broker message key SHOULD be the "jobId" value, enabling consumers to partition and deduplicate by job.

* The Content-Type of the message (if the broker supports content-type metadata) SHOULD be "application/problem+json".

Example context: A PDF generation service publishes a failure to a Kafka result topic. The consumer reads the Problem Details object from the message value.

5.3. Webhook Callbacks

When a server delivers a job result by POSTing to a client-supplied callback URL, the request body SHOULD be the problem details object with Content-Type "application/problem+json".

In this context:

* The HTTP status code of the webhook POST is about the delivery, not the job. The "status" member inside the problem details object conveys the job-level status.

* Servers SHOULD sign webhook deliveries using HTTP Message Signatures [RFC9421] to allow recipients to verify authenticity. See Section 6.6.

* The "retryAfter" member applies to job resubmission, NOT to webhook delivery retry. Webhook delivery retry is a concern of the delivery mechanism, not this specification.

5.4. Server-Sent Events

When a server pushes job status updates via Server-Sent Events (SSE) [W3C.SSE], the problem details object is serialized as the "data" field of an event.

The "event" field of the SSE SHOULD be "job-failed" (or a similar descriptive type).

Example:

```
event: job-failed
data: {"type":"https://api.example.com/problems/rendering-failed",
data:  "title":"Document Rendering Failed","status":500,
data:  "jobId":"550e8400","jobStatus":"FAILED",
data:  "processingStage":"rendering"}
```

6. Interaction with Existing Standards

This section documents how the extension members interact with each referenced RFC, to help implementers compose them correctly.

6.1. RFC 9457 — Problem Details for HTTP APIs

This document extends [RFC9457] by defining reusable extension members per Section 3.2 of that specification. All standard [RFC9457] members ("type", "title", "status", "detail", "instance") retain their original semantics.

Notably:

* "type" SHOULD be a URI identifying the specific failure type, NOT "about:blank", when async job extension members are present. This enables consumers to distinguish async job failures from generic HTTP errors.

* "instance" SHOULD identify the specific occurrence of the problem, per [RFC9457] Section 3.1.5. For async jobs, this is typically the URI of the job status resource (e.g., "/api/v1/documents/jobs/550e8400-e29b-41d4-a716-446655440000"), enabling consumers to dereference it for full details.

* "status" represents the HTTP status code that WOULD have been returned if the job had been processed synchronously.

6.2. RFC 9110 — HTTP Semantics (202 Accepted)

[RFC9110] Section 15.3.3 defines 202 Accepted as indicating that the request has been accepted for processing but the processing has not been completed. It notes that there is no facility in HTTP for re-connecting to an asynchronous operation later.

This document does not change the semantics of 202. The extension members defined here apply to the failure report, not to the acceptance response.

6.3. RFC 7240 — Prefer Header (respond-async)

[RFC7240] Section 4.1 defines the "respond-async" preference token. This document does not change its semantics.

When a server honors "respond-async" and later the job fails, the failure report (returned from the job status resource or delivered via another transport) SHOULD include the extension members defined in this document.

Note: The extension members defined here are useful regardless of whether the originating request included "Prefer: respond-async". A server that always processes requests asynchronously (without requiring the Prefer header) SHOULD still use these extension members when reporting job outcomes.

6.4. RFC 9512 — YAML Media Type

If an API documents its async job failure schemas using AsyncAPI [ASYNCAPI] specifications (commonly authored in YAML), the specification file SHOULD be served with Content-Type "application/yaml" per [RFC9512] when exposed via HTTP.

6.5. RFC 8288 — Web Linking

Servers that return 202 Accepted SHOULD include a "Link" header with a relation type that points to the job status resource. Servers MAY use the registered "status" relation type [RFC8631] or a URI-based extension relation type as described in Section 2.1.2 of [RFC8288].

When a completed or failed job’s status response includes a link to the result resource, the server SHOULD use a URI-based extension relation type as described in Section 2.1.2 of [RFC8288] (e.g., "https://api.example.com/rels/job-result") or an existing registered relation type such as "related" [IANA.LINK-RELATIONS]. Servers MUST NOT use unregistered short-form relation types.

6.6. RFC 9421 — HTTP Message Signatures

When job failure reports are delivered via webhooks, servers MAY sign the webhook request using [RFC9421] to allow the recipient to verify the report's authenticity and integrity.

When job results are delivered via message brokers, [RFC9421] does not apply (it is HTTP-specific). Broker-level security mechanisms (e.g., TLS, SASL) SHOULD be used instead.

6.7. RFC 6585 — Additional HTTP Status Codes (429)

[RFC6585] defines 429 Too Many Requests. This applies to the job status polling endpoint, NOT to job failure reports.

If a client polls too frequently, the server MAY return 429 with a "Retry-After" header. This is distinct from the "retryAfter" extension member, which governs job resubmission timing.

Implementers MUST NOT confuse these two retry signals:

* "Retry-After" header on a 429 response: "wait before polling again."

* "retryAfter" extension member in a FAILED job report: "wait before resubmitting the job."

6.8. RFC 9562 — UUIDs

The "jobId" member SHOULD be a UUID per [RFC9562] to ensure global uniqueness and unguessability. UUIDv7 is RECOMMENDED for new implementations because it embeds a timestamp, enabling natural chronological ordering of jobs.

6.9. RFC 6901 — JSON Pointer

When a job fails at the "validation" stage, the "detail" member MAY reference specific input fields using JSON Pointer [RFC6901] syntax (e.g., "Field /templateData/customerName is required").

For richer validation error reporting, implementers MAY combine these extension members with validation-specific extensions (such as a "violations" array using JSON Pointer field references), but defining such extensions is outside the scope of this document.

6.10. draft-ietf-httpapi-idempotency-key-header

When a client resubmits a failed job (guided by "retryable": true), it SHOULD include an "Idempotency-Key" header per [I-D.ietf-httpapi-idempotency-key-header] to prevent duplicate processing if the resubmission is received more than once.

The "jobId" from the failed job SHOULD NOT be reused as the idempotency key, because the retry is a new job submission.

6.11. draft-ietf-httpapi-ratelimit-headers

The RateLimit header fields defined in [I-D.ietf-httpapi-ratelimit-headers] apply to HTTP request rates for any endpoint, including job status polling endpoints.

The "retryAfter" extension member defined in this document applies to job resubmission intervals. These are orthogonal concerns.

7. Batch Operations

7.1. The "results" Extension Member

For batch operations that produce multiple outputs, this document defines the "results" extension member: an array of per-item outcome objects.

Each object in the array MUST contain:

* "itemId" (string): An identifier for the item within the batch. The value is server-defined and SHOULD correspond to an identifier from the original batch request. If the original batch request does not assign item identifiers, the server SHOULD assign them (e.g., zero-based array indices as strings: "0", "1", "2") and document the assignment scheme in the API specification.

* "status" (string): A terminal job status value from Section 4. Note: "COMPLETED_WITH_ERRORS" is not valid for individual batch items because a single item cannot itself be a partial batch; only the top-level batch "jobStatus" uses this value.

Each object MAY additionally contain:

* "detail" (string): A human-readable description of the outcome.

* "retryable" (boolean): Whether this specific item can be retried.

* "processingStage" (string): The stage at which this item failed.

7.1.1. Payload Size Considerations

For large batches (hundreds or thousands of items), including every item in the "results" array may produce an impractically large payload. Servers SHOULD apply the following strategies:

* Include only failed items in "results" when the batch jobStatus is "COMPLETED_WITH_ERRORS", and note the omission in the "detail" member (e.g., "3 of 5000 items failed; only failed items are listed").

* Define a maximum number of items in "results" (documented in the API specification) and provide a link to a paginated resource for the complete result set. Example:

  "detail": "150 of 10000 items failed. First 50 shown.",
  "results": ["... 50 items ..."]

  With a "Link" header or body link to the full result set.

* For very large batches, consider omitting "results" entirely and providing only a summary in "detail" with a link to the detailed results.

This document does not define a pagination mechanism for "results"; pagination is left to the API specification.

7.2. Partial Failure Semantics

When a batch contains both successes and failures:

* The "jobStatus" SHOULD be "COMPLETED_WITH_ERRORS".

* The "status" member (HTTP status equivalent) SHOULD be 207 (Multi-Status), as defined in [RFC4918] Section 13. Note that 207 was originally defined for WebDAV; its use in JSON batch APIs is a widely adopted industry convention.

* The "detail" member SHOULD summarize the outcome (e.g., "3 of 5 items completed successfully").

8. JSON Schema

The following JSON Schema [JSON-SCHEMA] defines the extension members introduced by this document. It is designed to be composed (via "allOf" or "$ref") with the Problem Details schema from Appendix A of [RFC9457].

```
   {
     "$schema": "https://json-schema.org/draft/2020-12/schema",
     "$id": "https://example.com/schemas/async-job-problem-details",
     "title": "Async Job Problem Details Extensions",
     "description": "Extension members for RFC 9457 Problem Details objects that describe
 asynchronous job outcomes.",
     "type": "object",
     "properties": {
       "jobId": {
         "type": "string",
         "description": "Unique identifier for the async job.",
         "examples": ["550e8400-e29b-41d4-a716-446655440000"]
       },
       "jobStatus": {
         "type": "string",
         "description": "Current state of the async job. Registered values are ACCEPTED,
PROCESSING, COMPLETED, FAILED, CANCELLED, TIMED_OUT, and COMPLETED_WITH_ERRORS. Servers M
AY define additional values per Section 4.3.",
         "examples": ["FAILED", "COMPLETED", "TIMED_OUT"]
       },
       "submittedAt": {
         "type": "string",
         "format": "date-time",
         "description": "RFC 3339 timestamp (UTC) when the job was accepted for processin
g.",
         "examples": ["2026-02-26T10:00:00Z"]
       },
       "completedAt": {
         "type": "string",
         "format": "date-time",
         "description": "RFC 3339 timestamp (UTC) when the job reached a terminal state."
,
         "examples": ["2026-02-26T10:00:05Z"]
       },
       "retryable": {
         "type": "boolean",
         "default": false,
         "description": "Whether the client should retry the job with the same input."
       },
       "retryAfter": {
         "type": "integer",
         "minimum": 0,
         "description": "Seconds to wait before retrying. Only meaningful when retryable
is true."
       },
       "processingStage": {
         "type": "string",
         "description": "The pipeline stage at which the failure occurred.",
         "examples": ["validation", "rendering", "conversion"]
       },
       "correlationId": {
         "type": "string",
         "description": "Client-supplied correlation identifier from the originating requ
est.",
         "examples": ["4bf92f3577b34da6a3ce929d0e0e4736"]
```

```
      },
      "results": {
        "type": "array",
        "description": "Per-item outcomes for batch operations.",
        "items": {
          "type": "object",
          "required": ["itemId", "status"],
          "properties": {
            "itemId": {
              "type": "string",
              "description": "Identifier for the batch item."
            },
            "status": {
              "type": "string",
              "enum": [
                "COMPLETED",
                "FAILED",
                "CANCELLED",
                "TIMED_OUT"
              ],
              "description": "Terminal outcome for this item."
            },
            "detail": {
              "type": "string",
              "description": "Human-readable outcome description."
            },
            "retryable": {
              "type": "boolean",
              "description": "Whether this item can be retried."
            },
            "processingStage": {
              "type": "string",
              "description": "Stage at which this item failed."
            }
          },
          "additionalProperties": true
        }
      }
    },
    "additionalProperties": true
  }
```

Note: The "jobStatus" property does not use a JSON Schema "enum" constraint because Section 4.3 allows servers to define additional status values. Validators that wish to restrict values to the registered set MAY add an "enum" constraint locally, but SHOULD accept unrecognized values gracefully in production.

9. Security Considerations

9.1. Information Exposure

The extension members defined here can reveal internal architecture details. Servers MUST consider what information is appropriate for their audience:

* "processingStage" reveals pipeline structure. For external-facing APIs, servers SHOULD use generic stage names ("processing", "conversion") rather than implementation-specific names ("kafka-consumer-group-rebalance", "s3-multipart-upload").

* "jobId" values MUST NOT encode sensitive data (e.g., user IDs, database primary keys).

* "detail" messages MUST NOT include stack traces, internal hostnames, database queries, or file system paths.

9.2. Correlation ID Injection

The "correlationId" is client-supplied and MUST be treated as untrusted input.

* Servers MUST sanitize the value before including it in any output context -- log files, HTTP headers, HTML pages, database queries, or template rendering -- to prevent injection attacks including log injection (newline injection, ANSI escape sequences), HTTP header injection (CRLF injection), and cross-site scripting (XSS).

* Servers SHOULD enforce a maximum length (e.g., 256 characters) and restrict the character set (e.g., alphanumeric, hyphens, underscores, periods).

* Servers MUST NOT use the correlation ID in SQL queries, template rendering, or any context where injection is possible, without proper escaping.

9.3. Job Identifier Enumeration

If job identifiers are sequential or predictable, an attacker can enumerate job status resources belonging to other clients.

* Servers MUST use unguessable identifiers. UUIDv4 or UUIDv7 [RFC9562] are RECOMMENDED.

* Servers SHOULD enforce authorization on job status resources: a client MUST only be able to access jobs it submitted.

* Servers SHOULD implement rate limiting on the job status endpoint to slow enumeration attempts.

9.4. Retry Amplification

A malicious or compromised server could return "retryable": true with "retryAfter": 0 to induce clients to retry in a tight loop, creating a self-inflicted denial of service.

Clients MUST enforce:

* A maximum retry count (e.g., 5 attempts).

* A minimum backoff floor (e.g., 1 second), regardless of the server's "retryAfter" value. A "retryAfter" value of 0 MUST NOT be interpreted as "retry immediately".

* Exponential backoff with jitter for successive retries.

9.5. Timing Side Channels

The "submittedAt" and "completedAt" timestamps reveal processing duration. In sensitive contexts, this could leak information about:

* Server load (longer processing times indicate high load).

* Input complexity (more complex inputs may take longer).

* Conditional branching (different code paths take different amounts of time).

Servers operating in high-security environments MAY omit these members or round them to reduce precision.

9.6. Batch Results Information Scoping

In multi-tenant systems where a batch may contain items belonging to different authorization domains, the "results" array MUST only include items that the requesting client is authorized to view. Servers MUST NOT leak information about other tenants' items through the "results" array, the "detail" summary, or the item count.

9.7. Privacy Considerations

The extension members defined here may constitute or contain personally identifiable information (PII):

* "jobId" values, if correlated with external data, could identify individual users or transactions.

* "correlationId" values are client-supplied and may contain user identifiers, session tokens, or other PII.

* "submittedAt" and "completedAt" timestamps can reveal usage patterns that may be attributable to individuals.

Servers SHOULD evaluate the privacy implications of including these members in responses that may be logged, cached, or forwarded through intermediaries. In jurisdictions with data protection regulations (e.g., GDPR, CCPA), operators SHOULD ensure that problem details objects containing PII are treated as personal data for retention and access control purposes.

Servers SHOULD NOT include these extension members in responses that are cacheable or served to unauthenticated clients unless the information is already public.

10. IANA Considerations

10.1. Problem Details Extension Members

This document defines extension members for [RFC9457] problem details objects per the extension mechanism in Section 3.2 of [RFC9457]. As [RFC9457] does not establish a registry for extension members, the members defined in this document ("jobId", "jobStatus", "submittedAt", "completedAt", "retryable", "retryAfter", "processingStage", "correlationId", and "results") are specified here and do not require IANA registration.

Implementers that define additional extension members for async job problem details SHOULD choose names that do not conflict with the members defined in this document.

10.2. Job Status Value Registry

This document requests that IANA create a "Problem Details Async Job Status Values" registry with the initial entries from Table 4 (Section 4.1).

Registration policy: Specification Required [RFC8126].

Each registration MUST include:

* Status Value: An UPPER_SNAKE_CASE string (e.g., "PAUSED").

* Terminal: Whether this status is terminal (yes/no).

* Description: A brief description of the status meaning.

* Reference: A reference to the specification defining the status value.

The designated expert(s) SHOULD verify that proposed values use UPPER_SNAKE_CASE, do not duplicate the semantics of existing registered values, and are clearly documented as either terminal or non-terminal.

10.3. Problem Type URIs

This document does not register any problem type URIs. The "type" member in the examples uses illustrative URIs under the "api.example.com" domain.

Servers SHOULD define their own problem type URIs under a domain they control. When async job extension members are present, the "type" member SHOULD NOT be "about:blank"; a specific URI helps consumers distinguish async job failures from generic HTTP errors. See [RFC9457] Section 4.2.1 for guidance on using "about:blank".

11. Examples

All examples are complete problem details objects that validate against the schema in Section 8.

11.1. HTTP Response: Document Rendering Failure

A document generation API processed a PDF request asynchronously. The rendering stage failed. The client discovers this by polling the job status resource.

HTTP context:

```
GET /api/v1/documents/jobs/550e8400-e29b-41d4-a716-446655440000 HTTP/1.1
Host: api.example.com

HTTP/1.1 200 OK
Content-Type: application/problem+json
```

```
{
  "type": "https://api.example.com/problems/rendering-failed",
  "title": "Document Rendering Failed",
  "status": 500,
  "detail": "Template 'invoice-v2' contains an unclosed element at line 87",
  "instance": "/api/v1/documents/jobs/550e8400-e29b-41d4-a716-446655440000",
  "jobId": "550e8400-e29b-41d4-a716-446655440000",
  "jobStatus": "FAILED",
  "submittedAt": "2026-02-26T10:00:00Z",
  "completedAt": "2026-02-26T10:00:03Z",
  "retryable": false,
  "processingStage": "rendering",
  "correlationId": "4bf92f3577b34da6a3ce929d0e0e4736"
}
```


Note: The HTTP status is 200 (the status check succeeded). The "status": 500 inside the object indicates the equivalent synchronous failure code. This distinction is intentional: the client's request to check the job status was successful (HTTP 200); the job itself failed (Problem Details status 500). Client libraries that interpret "application/problem+json" as an error signal SHOULD inspect the HTTP status code first; a 200 response with Problem Details content indicates a successfully retrieved failure report, not a request failure.

11.2. HTTP Response: Job Timeout with Retry

A report generation job exceeded the 300-second limit. The server indicates the failure is transient and suggests retrying after 60 seconds.


```
{
  "type": "https://api.example.com/problems/job-timed-out",
  "title": "Job Processing Timed Out",
  "status": 504,
  "detail": "Job exceeded maximum processing time of 300s",
  "jobId": "7c9e6679-7425-40de-944b-e07fc1f90ae7",
  "jobStatus": "TIMED_OUT",
  "submittedAt": "2026-02-26T09:00:00Z",
  "completedAt": "2026-02-26T09:05:00Z",
  "retryable": true,
  "retryAfter": 60,
  "processingStage": "processing"
}
```

11.3. Kafka Message: PDF Conversion Failure

A PDF generation service publishes a failure to a Kafka result topic. No HTTP headers are available; the "retryAfter" member (absent here because retryable is false) would be the sole mechanism for conveying retry guidance if present.

Kafka message:

```
Topic:   com.example.pdf-job.result.v1
Key:     d4735e3a-265e-16d0-8f24-2de10e933e80
Headers: content-type=application/problem+json
Value:
```

```
{
  "type": "https://api.example.com/problems/conversion-failed",
  "title": "PDF Conversion Failed",
  "status": 502,
  "detail": "iText HTML-to-PDF conversion failed: malformed CSS at line 15",
  "jobId": "d4735e3a-265e-16d0-8f24-2de10e933e80",
  "jobStatus": "FAILED",
  "submittedAt": "2026-02-26T16:30:00Z",
  "completedAt": "2026-02-26T16:30:02Z",
  "retryable": false,
  "processingStage": "conversion",
  "correlationId": "monthly-report-2026-02"
}
```

Note: The "retryAfter" member is absent because the failure is not retryable. The "status": 502 indicates the equivalent HTTP status; it is not an HTTP response code in this context.

11.4. Webhook Callback: Export Completed with Errors

A data export service delivers a batch result via webhook.

```
POST /webhooks/export-results HTTP/1.1
Host: client.example.com
Content-Type: application/problem+json
Signature: sig1=:BASE64SIGNATURE:
Signature-Input: sig1=("content-type" "content-digest");keyid="server-key-1";created=1
740567600
```

```
{
  "type": "https://api.example.com/problems/batch-partial",
  "title": "Data Export Partially Failed",
  "status": 207,
  "detail": "8 of 10 records exported successfully",
  "jobId": "export-batch-20260226",
  "jobStatus": "COMPLETED_WITH_ERRORS",
  "submittedAt": "2026-02-26T12:00:00Z",
  "completedAt": "2026-02-26T12:15:00Z",
  "results": [
    {
      "itemId": "rec-007",
      "status": "FAILED",
      "detail": "Record exceeds maximum size (10MB)",
      "retryable": false
    },
    {
      "itemId": "rec-009",
      "status": "FAILED",
      "detail": "Downstream storage timeout",
      "retryable": true,
      "processingStage": "storage"
    }
  ]
}
```

Note: Only failed items are included in "results" to reduce payload size. The webhook request itself uses HTTP Message Signatures [RFC9421] for authenticity.

11.5. Server-Sent Event: Processing Stage Update

A Server-Sent Event stream delivers a failure notification.

```
event: job-failed
id: 550e8400-e29b-41d4-a716-446655440000
data: {"type":"https://api.example.com/problems/rendering-failed",
data:  "title":"Document Rendering Failed","status":500,
data:  "jobId":"550e8400","jobStatus":"FAILED",
data:  "submittedAt":"2026-02-26T10:00:00Z",
data:  "completedAt":"2026-02-26T10:00:03Z",
data:  "retryable":false,"processingStage":"rendering"}
```

11.6. Batch: Partial Failure

Three certificate generation requests; two succeed, one fails.

```
{
  "type": "https://api.example.com/problems/batch-partial",
  "title": "Batch Processing Partially Failed",
  "status": 207,
  "detail": "2 of 3 certificates generated successfully",
  "jobId": "batch-a1b2c3d4",
  "jobStatus": "COMPLETED_WITH_ERRORS",
  "submittedAt": "2026-02-26T14:00:00Z",
  "completedAt": "2026-02-26T14:00:12Z",
  "results": [
    {
      "itemId": "cert-001",
      "status": "COMPLETED"
    },
    {
      "itemId": "cert-002",
      "status": "FAILED",
      "detail": "Required field 'recipientName' missing",
      "retryable": false,
      "processingStage": "validation"
    },
    {
      "itemId": "cert-003",
      "status": "COMPLETED"
    }
  ]
}
```

11.7. HTTP Response: Successful Job Completion

The extension members are not limited to failure reporting. A server MAY include them in a successful job status response to provide timing and identification context.

```
GET /api/v1/documents/jobs/a1b2c3d4-5678-90ab-cdef-1234567890ab HTTP/1.1
Host: api.example.com

HTTP/1.1 200 OK
Content-Type: application/json
Link: <https://api.example.com/api/v1/documents/results/a1b2c3d4>;
      rel="https://api.example.com/rels/job-result"
```

```
{
  "jobId": "a1b2c3d4-5678-90ab-cdef-1234567890ab",
  "jobStatus": "COMPLETED",
  "submittedAt": "2026-02-26T10:00:00Z",
  "completedAt": "2026-02-26T10:00:04Z",
  "correlationId": "invoice-batch-2026-02-26"
}
```

Note: The Content-Type is "application/json", not "application/problem+json", because this is not an error response. The extension members defined in this document are JSON members that MAY appear in any JSON object; they are not restricted to [RFC9457] problem details objects. However, when reporting failures, "application/problem+json" SHOULD be used per [RFC9457].

11.8. Kafka Message: Transient Failure with Retry Guidance

A report generation service publishes a transient failure to a Kafka result topic, demonstrating the transport-independence value of the "retryAfter" member.

Kafka message:

```
Topic:   com.example.report-job.result.v1
Key:     e5f6a7b8-9012-3456-7890-abcdef123456
Headers: content-type=application/problem+json
Value:
```

```
{
  "type": "https://api.example.com/problems/downstream-unavailable",
  "title": "Downstream Service Temporarily Unavailable",
  "status": 503,
  "detail": "Data warehouse connection pool exhausted; service is expected to recover within 60 seconds",
  "jobId": "e5f6a7b8-9012-3456-7890-abcdef123456",
  "jobStatus": "FAILED",
  "submittedAt": "2026-02-26T18:00:00Z",
  "completedAt": "2026-02-26T18:00:01Z",
  "retryable": true,
  "retryAfter": 60,
  "processingStage": "processing",
  "correlationId": "weekly-report-2026-w09"
}
```

Note: In an HTTP context, the server would also include a "Retry-After: 60" header. In this Kafka context, the "retryAfter" member is the only way to convey the retry interval to the consumer.

12. References

12.1. Normative References

[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/info/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/info/rfc3339>.

[RFC7240] Snell, J., "Prefer Header for HTTP", RFC 7240, DOI 10.17487/RFC7240, June 2014, <https://www.rfc-editor.org/info/rfc7240>.

[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/info/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/info/rfc8259>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/info/rfc9457>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

12.2. Informative References

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <https://www.rfc-editor.org/info/rfc4918>.

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <https://www.rfc-editor.org/info/rfc6585>.

[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/info/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/info/rfc8126>.

[RFC8631] Wilde, E., "Link Relation Types for Web Services", RFC 8631, DOI 10.17487/RFC8631, July 2019, <https://www.rfc-editor.org/info/rfc8631>.

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/info/rfc9421>.

[RFC9512] Wilde, E. and R. Amsss, "YAML Media Type", RFC 9512, DOI 10.17487/RFC9512, February 2024, <https://www.rfc-editor.org/info/rfc9512>.

[JSON-SCHEMA] Wright, A., Andrews, H., Hutton, B., and G. Dennis, "JSON Schema: A Media Type for Describing JSON Documents", Work in Progress, Internet-Draft, draft-bhutton-json-schema-01, June 2022, <https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-01>.

[ASYNCAPI] AsyncAPI Initiative, "AsyncAPI Specification", Version 3.0.0, February 2024, <https://www.asyncapi.com/docs/reference/specification/v3.0.0>.

[W3C.TRACE-CONTEXT] Rodrigues, W., Kanzhelev, S., Mace, J., and B. Banerjee, "Trace Context", W3C Recommendation, November 2021, <https://www.w3.org/TR/trace-context/>.

[W3C.SSE] Hickson, I., "Server-Sent Events", W3C Recommendation, February 2015, <https://www.w3.org/TR/eventsource/>.

[I-D.ietf-httpapi-idempotency-key-header]
Dalal, S. and J. Jena, "The Idempotency-Key HTTP Header Field", Work in Progress, Internet-Draft, draft-ietf-httpapi-idempotency-key-header, <https://datatracker.ietf.org/doc/html/draft-ietf-httpapi-idempotency-key-header>.

[I-D.ietf-httpapi-ratelimit-headers]
Polli, R. and A. Martinez, "RateLimit header fields for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpapi-ratelimit-headers, <https://datatracker.ietf.org/doc/html/draft-ietf-httpapi-ratelimit-headers>.

[GOOGLE-AIP-151]
Google, "AIP-151: Long-running operations", <https://google.aip.dev/151>.

[CLOUDEVENTS]
Cloud Native Computing Foundation, "CloudEvents Specification", Version 1.0.2, <https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/spec.md>.

[OPENAPI] OpenAPI Initiative, "OpenAPI Specification", Version 3.1.0, <https://spec.openapis.org/oas/v3.1.0>.

[IANA.LINK-RELATIONS]
IANA, "Link Relations", <https://www.iana.org/assignments/link-relations/link-relations.xhtml>.

Appendix A. Comparison with Industry Patterns

This appendix provides an expanded comparison of async job error reporting across platforms, illustrating the fragmentation that this specification addresses.

A.1. AWS Step Functions

AWS Step Functions represent failures as:

```
{
  "executionArn": "arn:aws:states:...",
  "status": "FAILED",
  "error": "States.TaskFailed",
  "cause": "Lambda function threw an exception",
  "startDate": "2026-02-26T10:00:00.000Z",
  "stopDate": "2026-02-26T10:00:05.000Z"
}
```

Mapping to this specification: "executionArn" → "jobId", "status" → "jobStatus", "error"+"cause" → "detail", "startDate" → "submittedAt", "stopDate" → "completedAt". No retry guidance. No processing stage. Not RFC 9457.

A.2. Google AIP-151 (Long-Running Operations)

Google uses a Protobuf-based "Operation" resource:

```
{
  "name": "operations/abc123",
  "done": true,
  "error": {
    "code": 3,
    "message": "Invalid template",
    "details": []
  }
}
```

Mapping: "name" → "jobId", "done" → terminal state check, "error" → google.rpc.Status (not RFC 9457). No "submittedAt"/"completedAt". No retry guidance. No processing stage.

A.3. Azure Long-Running Operations

Azure uses a polling pattern with:

```
{
  "id": "job-xyz",
  "status": "Failed",
  "error": {
    "code": "RenderingFailed",
    "message": "Template syntax error at line 42"
  },
  "startTime": "2026-02-26T10:00:00Z",
  "endTime": "2026-02-26T10:00:05Z"
}
```

Mapping: "id" → "jobId", "status" → "jobStatus", "error" → "detail", "startTime" → "submittedAt", "endTime" → "completedAt". "retryAfter" sometimes in HTTP header only. No processing stage. Not RFC 9457.

A.4. Stripe

Stripe uses:

```
{
  "id": "pi_abc123",
  "status": "failed",
  "last_payment_error": {
    "type": "card_error",
    "message": "Your card was declined"
  },
  "created": 1740567600
}
```

Mapping: "id" → "jobId", "status" → "jobStatus", "last_payment_error" → "detail", "created" → "submittedAt". No completion time. No retry guidance. No processing stage. Unix timestamp, not RFC 3339. Not RFC 9457.

Appendix B. Design Decisions

B.1. Why Extension Members, Not a New Media Type?

Defining a new media type (e.g., "application/async-job-problem+json") was considered and rejected because:

* [RFC9457] already provides a well-known envelope with Content-Type negotiation ("application/problem+json").

* Extension members integrate seamlessly with existing [RFC9457] infrastructure (exception mappers, middleware, logging).

* Clients that don’t understand the extensions can still process the standard Problem Details members ("type", "title", "status", "detail").

B.2. Why JSON Members for Retry Instead of Only Headers?

The "Retry-After" HTTP header field [RFC9110] already exists. The "retryAfter" JSON member was added because:

* Message broker payloads have no HTTP headers.

* Webhook callbacks use HTTP, but the "Retry-After" header applies to the webhook delivery, not the job retry.

* The JSON member travels with the Problem Details object across all transports without transformation.

B.3. Why an Explicit "retryable" Boolean?

The presence of "retryAfter" alone could imply retryability. A separate boolean was added because:

* Explicitness aids machine readability — no inference needed.

* Some failures are retryable without a specific delay recommendation ("retryable": true without "retryAfter").

* Some APIs may want to convey "retryAfter" for general backoff purposes even on non-retryable errors (Section 3.7 allows this in exceptional cases); the explicit boolean prevents ambiguity about whether a retry is advised.

B.4. Why "processingStage" Is a String, Not an Enum?

Processing pipelines vary too widely across APIs to define a fixed enum. A free-form string with recommended values (Table 3) provides flexibility while encouraging consistency through convention.

B.5. Why "correlationId" References W3C Trace Context?

The W3C Trace Context specification [W3C.TRACE-CONTEXT] is the de facto standard for distributed tracing headers. Referencing it provides interoperability with OpenTelemetry, Jaeger, Zipkin, and other tracing systems without requiring this specification to define its own tracing format.

Appendix C. AsyncAPI Integration

APIs that use AsyncAPI [ASYNCAPI] to document message-driven interfaces can reference this specification in their schemas.

Example AsyncAPI schema fragment:

```
components:
  schemas:
    PdfJobFailedEvent:
      description: >
        PDF generation failure report per
        draft-ratnawat-httpapi-async-problem-details.
      allOf:
        - $ref: '#/components/schemas/ProblemDetails'
        - type: object
          required: [jobId, jobStatus, submittedAt]
          properties:
            jobId:
              type: string
              format: uuid
            jobStatus:
              type: string
              enum: [FAILED, TIMED_OUT, CANCELLED]
            submittedAt:
              type: string
              format: date-time
            completedAt:
              type: string
              format: date-time
            retryable:
              type: boolean
            retryAfter:
              type: integer
              minimum: 0
            processingStage:
              type: string
            correlationId:
              type: string
```

Appendix D. Relationship to CloudEvents

CloudEvents [CLOUDEVENTS] is a CNCF specification for describing events in a standard way. It defines context attributes including "id", "source", "type", and "subject" that overlap conceptually with some members defined here:

* CloudEvents "id" "jobId" (event/job identifier)

* CloudEvents "source" "instance" (origin of the event)

* CloudEvents "time" "completedAt" (timestamp)

The specifications are complementary, not competing:

* CloudEvents defines the event envelope (metadata about the event itself: what happened, where, when).

* This specification defines the event payload content (structured failure context: what failed, at what stage, whether to retry).

When using CloudEvents as the transport envelope for async job failure notifications, the problem details object (with the extension members defined here) SHOULD be the CloudEvents "data" payload. The CloudEvents "datacontenttype" SHOULD be "application/problem+json".

Example CloudEvents + Problem Details:

```
{
  "specversion": "1.0",
  "id": "evt-550e8400",
  "source": "/api/v1/documents/generate",
  "type": "com.example.job.failed",
  "datacontenttype": "application/problem+json",
  "data": {
    "type": "https://api.example.com/problems/rendering-failed",
    "title": "Document Rendering Failed",
    "status": 500,
    "jobId": "550e8400-e29b-41d4-a716-446655440000",
    "jobStatus": "FAILED",
    "processingStage": "rendering",
    "retryable": false
  }
}
```

Appendix E. OpenAPI Integration

APIs documented with the OpenAPI Specification [OPENAPI] can reference the JSON Schema defined in Section 8 to describe async job failure responses.

Example OpenAPI schema fragment (YAML):

```
components:
  schemas:
    AsyncJobProblemDetails:
      description: >
        RFC 9457 Problem Details extended with async job context
        per draft-ratnawat-httpapi-async-problem-details.
      allOf:
        - $ref: '#/components/schemas/ProblemDetails'
        - $ref: 'https://example.com/schemas/async-job-problem-details'

paths:
  /api/v1/jobs/{jobId}:
    get:
      summary: Check async job status
      responses:
        '200':
          description: >
            Job status. Content-Type is application/json for
            non-terminal and COMPLETED states, and
            application/problem+json for failed states.
          content:
            application/problem+json:
              schema:
                $ref: '#/components/schemas/AsyncJobProblemDetails'
```

Appendix F. Interoperability Considerations

F.1. Handling Partial Member Sets

A client that receives a problem details object with some but not all of the extension members defined here SHOULD process the members that are present and ignore the absence of others. For example, a response with "jobId" and "jobStatus" but without "submittedAt" is valid and useful.

F.2. Handling Unknown Extension Members

Per [RFC9457] Section 3.2, consumers MUST ignore extension members they do not understand. This ensures forward compatibility: future revisions of this document may define additional members without breaking existing clients.

F.3. Handling Unrecognized Job Status Values

As specified in Section 4.3, clients that encounter an unrecognized "jobStatus" value SHOULD treat it as non-terminal (equivalent to "PROCESSING"). This provides graceful degradation when a server uses custom status values.

F.4. XML Representation

[RFC9457] defines both JSON and XML representations for problem details. This document defines extension members for the JSON representation only. Servers that need to produce XML problem details objects may map the extension members to XML elements using the conventions in [RFC9457] Section 4, but this document does not define a normative XML mapping.

F.5. Internationalization of "detail"

The "detail" member is human-readable text. Per [RFC9457] Section 3.1.4, the language of "detail" and "title" is determined by Content-Language negotiation.

Servers that serve multilingual audiences SHOULD respect the "Accept-Language" header from the originating request when generating the "detail" text for asynchronous failure reports. When the failure report is delivered via a non-HTTP transport where language negotiation is not available, servers SHOULD use English (en) as the default language and MAY include a "Content-Language" metadata field if the transport supports it.

Acknowledgements

The extension members defined in this document were informed by real-world experience building production asynchronous document generation services that process requests via Apache Kafka. The challenge of reporting structured errors across synchronous HTTP and asynchronous messaging boundaries motivated this work.

The author thanks the IETF HTTP API Working Group (httpapi) for their ongoing work on [RFC9457], [I-D.ietf-httpapi-idempotency-key-header], and [I-D.ietf-httpapi-ratelimit-headers] -- which collectively address the mechanics of async API patterns and provided the foundation for this complementary specification.

Google’s AIP-151 (Long-Running Operations) [GOOGLE-AIP-151] influenced the status value design, though this specification chose RFC 9457 as the envelope rather than gRPC/Protobuf.

The CloudEvents specification [CLOUDEVENTS] and the OpenAPI Specification [OPENAPI] informed the transport-independence design principle and the integration guidance in the appendices.

Author’s Address

Gaurav Ratnawat
IMTF
Email: gaurav.ratnawat@imtf.com