API Design Rules Module: Transfer

Logius Guide
Draft August 07, 2026

This version:
https://logius-standaarden.github.io/API-mod-transfer/

Latest published version:
https://gitdocumentatie.logius.nl/publicatie/api/mod-transfer/

Latest editor's draft:
https://logius-standaarden.github.io/API-mod-transfer/

Previous version:
https://gitdocumentatie.logius.nl/publicatie/api/mod-transfer//

Editor:
Logius Standaarden (Logius)

Author:
Logius Standaarden (Logius)

Participate:
GitHub Logius-standaarden/API-mod-transfer
All issues
File an issue
Commit history
Pull requests

This document is also available in these non-normative format: PDF

Status of This Document

This is a draft that could be altered, removed or replaced by other documents. It is not a recommendation approved by TO.

Table of Contents

§Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, and *SHOULD* 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.

Abstract

This document is part of the *Nederlandse API Strategie*.

The Nederlandse API Strategie consists of a set of distinct documents.

Status	Description & Link
Informative	Inleiding NL API Strategie
Informative	Architectuur NL API Strategie
Informative	Gebruikerswensen NL API Strategie
Normative	NLgov REST API Design Rules (ADR v2.2)
Normative	Open API Specification (OAS 3.0)
Normative	NLgov Assurance profile for OAuth 2.0
Normative	Digikoppeling REST API koppelvlak specificatie
Normative module	GEO module v1.0

Before reading this document it is advised to gain knowledge of the informative documents, in particular the Architecture.

This document describes the *Transfer module*, containing rules for transferring large files.

1. Introduction

This document is a module as part of the *API Design Rules*. It is based on *Digikoppeling Koppelvlakstandaard Grote Berichten* which was initially written for ebMS2 and WUS-based exchanges in Digikoppeling.

Resources in REST APIs are usually represented as a JSON or XML body. Some payloads are not suitable for such a format, such as media files or bulk data exports. This module provides rules for describing such files as a metadata resource and transferring them reliably using two patterns: pull and push.

1.1 Pull pattern

The sender (server) makes the file available at a dedicated URI (`contentUri`). The receiver (client) fetches the file with `GET`. Interrupted transfer can be resumed.

1.2 Push pattern

The sender (client) requests an upload URI from the receiver (server), using `POST`. Subsequently, the sender uploads the file to the provided URI (`contentUri`), using `PUT`.

2. Design rules

Summary

Design rules are technical rules, which should be tested automatically, and functional rules, which should be considered when designing and building the API.

List of technical rules

- /transfer/metadata: Describe a large file using a metadata resource
- /transfer/range: Support HTTP range requests when retrieving large files
- /transfer/integrity: Use integrity fields to verify transfers
- /transfer/upload: Request an upload location, then use HTTP PUT to push

List of functional rules

- /transfer/threshold: Agree on a size threshold above which this module applies

Rules

Functional

/transfer/threshold: Agree on a size threshold above which this module applies

Statement

Sender and receiver *MAY* agree bilaterally on a payload size threshold above which this module applies. In lieu of such an agreement, this module *SHOULD* be applied for payloads over 20 MiB.

Rationale

Size limits depend on the infrastructure of the sender and receiver. Agreeing on a limit is therefore preferred. A default is provided to not force a negotiation for each integration.

Technical

/transfer/metadata: Describe a large file using a metadata resource

Statement

A large file *MUST* be described as a JSON ***metadata resource*** containing: `fileName`, `contentType`, `size` (bytes), and `contentUri`. It *MAY* also contain `createdAt` and `expiresAt`.

When receiving a file, the API *MUST* verify that the size of the received content matches `size`. A mismatch *MUST* be treated as an unsuccessful transfer.

EXAMPLE 1: Metadata resource

```
GET /files/80444340-6d5b-4e6c-8192-b1b935502790 HTTP/1.1
```

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

{
  "fileName": "file.pdf",
  "contentType": "application/pdf",
  "size": 2147483648,
  "createdAt": "2026-01-01T09:00:00Z",
  "expiresAt": "2026-01-15T09:00:00Z",
  "contentUri": "https://api.example.org/v1/files/80444340
}
```

Rationale

A receiver needs to know what it is about to transfer so it can commit storage, bandwidth, and processing time. The optional availability window avoids transfer attempts for content that has been removed.

How to test

1. Retrieve the metadata resource and validate the presence of the required fields.
2. Transfer content to a push `contentUri` with an incorrect `size` value and verify it is rejected.

Technical

/transfer/range: Support HTTP range requests when retrieving large files

Statement

When presenting a file for retrieval, the API *MUST* support range requests with the range unit set to `bytes`, as defined in [*RFC9110*] (section 14). This *MUST* be indicated with `Accept-Ranges: bytes` in the response header.

The response header *MUST* also include a strong entity-tag (`ETag`), so the receiver can reliably resume the transfer with `If-Range`.

EXAMPLE 2: Pull with range request

```
GET /files/80444340-6d5b-4e6c-8192-b1b935502790/content HT
Range: bytes=1048576-
If-Range: "8f2c1b4e9a"
```

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1048576-2147483647/2147483648
Accept-Ranges: bytes
ETag: "8f2c1b4e9a"
Content-Digest: sha-256=:Fw8SCU41fk3m26+GAXStg+us/0SflGTjb
Repr-Digest: sha-256=:o0vGm8t1aMYw54e3f7zFEl9CDkEDpMl9fGG4
```

NOTE

This rule only applies to retrieval, even though RFC 9110 mentions partial `PUT` via `Content-Range`. According to the Internet-Draft *Resumable Uploads for HTTP*: "there are caveats that affect its deployability". The aforementioned draft offers a method for resumable uploads. It is a work in progress and therefore not included as a rule in this module.

Rationale

Large transfers take longer and are therefore more likely to be interrupted. With range support, the transfer can be resumed without having the previously received data go to waste.

How to test

1. Send a `GET` request without a `Range` header.
   - Validate the response is `200` and contains the `Accept-Ranges: bytes` and `ETag` headers.
2. Send a `GET` request with a `Range` header.
   - Validate the response is `206` and with a correct `Content-Range` header.
3. Send a `GET` request with a `Range` header and an `If-Range` that does not match the `ETag`.
   - Validate the response is `200` instead of `206`.

/transfer/integrity: Use integrity fields to verify transfers

Technical

Statement

Requests and responses transferring large files (to or from `contentUri`) *MUST* include `Content-Digest` [*RFC9530*] with `sha-256` or stronger.

Responses transferring large files *MUST* also include `Repr-Digest` [*RFC9530*].

When receiving a file, the API *MUST* compute the digest of the received content and verify it matches `Content-Digest`. A mismatch *MUST* be treated as an unsuccessful transfer.

> NOTE
>
> A response may contain only a fragment of the file (see /transfer/range), in which case `Content-Digest` covers only the content of that message. `Repr-Digest` covers the complete file, regardless of range used.

EXAMPLE 3: Pull with integrity fields

```
GET /files/80444340-6d5b-4e6c-8192-b1b935502790/content HT
```

```
HTTP/1.1 200 OK
Content-Type: application/pdf
Accept-Ranges: bytes
ETag: "8f2c1b4e9a"
Content-Digest: sha-256=:o0vGm8t1aMYw54e3f7zFEl9CDkEDpMl9f
Repr-Digest: sha-256=:o0vGm8t1aMYw54e3f7zFEl9CDkEDpMl9fGG4
```

Rationale

Large transfers are more likely to be corrupted. Validating the arrived content alleviates this concern.

If the file is retrieved in parts, the receiver needs a digest that covers the whole file instead of a part.

How to test

1. Verify the response of a pull `contentUri` contains `Content-Digest` and `Repr-Digest` using `sha-256` or stronger.
2. Transfer content to a push `contentUri` with an incorrect `Content-Digest` value and verify it is rejected.

Technical

/transfer/upload: Request an upload location, then use HTTP PUT to push

Statement

An API accepting large files *MUST* publish an endpoint to which the sender can `POST` a partial metadata resource containing `fileName`, `contentType`, `size`. Upon acceptance, the API *MUST* respond with `201 Created` with the `Location` header pointing to the metadata resource and a body with the completed metadata resource.

The API *MUST* support HTTP `PUT` requests to the `contentUri`.

EXAMPLE 4: Push exchange

```
POST /files HTTP/1.1
Content-Type: application/json

{
  "fileName": "file.pdf",
  "contentType": "application/pdf",
  "size": 2147483648
}
```

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://api.example.org/v1/files/80444340-6d5b-4
```

```
{
  "fileName": "file.pdf",
  "contentType": "application/pdf",
  "size": 2147483648,
  "createdAt": "2026-01-01T09:00:00Z",
  "expiresAt": "2026-01-15T09:00:00Z",
  "contentUri": "https://api.example.org/v1/files/80444340
}
```

```
PUT /files/80444340-6d5b-4e6c-8192-b1b935502790/content HT
Content-Type: application/pdf
Content-Digest: sha-256=:o0vGm8t1aMYw54e3f7zFEl9CDkEDpMl9f
```

```
HTTP/1.1 204 No Content
```

Rationale

The receiver should be the one to decide the location of a large file sent to their server. Through `POST` the sender can request a location in an automated fashion. This also gives the receiver a chance to refuse the transfer based on content type or size before time and bandwidth are wasted.

Large transfers take longer and are therefore more likely to be interrupted. Resending the file should not create another file, but overwrite the previous attempt. That's why `PUT` with its idempotent nature is chosen over `POST` for the file transfer.

How to test

1. Validate `contentUri` accepts `PUT`.
2. Verify a `201` response is defined including a `Location` header and a body in the metadata resource format.

A. References

A.1 Normative references

[ADR]
API Design Rules. Jasper Roes; Joost Farla. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/api/adr/

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/info/rfc2119/

[RFC8174]
Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/info/rfc8174/

[RFC9110]
HTTP Semantics. R. Fielding, Ed.; M. Nottingham, Ed.; J. Reschke, Ed. IETF. June 2022. Internet Standard. URL: https://httpwg.org/specs/rfc9110.html

[RFC9530]
Digest Fields. R. Polli; L. Pardue. IETF. February 2024. Proposed Standard. URL: https://www.rfc-editor.org/info/rfc9530/

§A.2 Informative references

[DK-GB]

Digikoppeling Koppelvlakstandaard Grote Berichten. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/dk/gb/

[httpbis-resumable-upload]

Resumable Uploads for HTTP. Marius Kleidl; Guoye Zhang; Lucas Pardue. IETF. 2026-07-06. Active draft. URL: https://datatracker.ietf.org/doc/draft-ietf-httpbis-resumable-upload/

↥