Developer Guide

Developer documentation for extending and maintaining the system.

Patient Record Import API

Open Clinical History provides a REST-style API for submitting patient history documents directly into the clinical-history processing pipeline.

The API endpoint is:

```
history_api_import.php
```

For example:

```
https://www.openclinicalhistory.org/demo/history_api_import.php
```

The API can:

- create or locate a patient
- receive unstructured clinical history text
- queue the document for extraction
- process the document through the Open Clinical History extraction pipeline
- monitor processing status
- automatically audit and commit safe clinical events
- manually initiate the audit/commit stage
- inspect and manage the ingest queue

The API uses the **same clinical processing pipeline and safety rules as the interactive patient-history importer**.

API Processing Flow

A typical API import follows this sequence:

```
External Clinical System
        |
        | POST patient + document
        v
history_api_import.php
        |
```

```
        v
Find/Create Patient
        |
        v
Store Source Document
        |
        v
Ingest Queue
        |
        v
queue_worker.php
        |
        v
Clinical Extraction
        |
        v
SNOMED Resolution
        |
        v
Anatomical Mapping
        |
        v
Clinical Audit
        |
        +--------------------------+
        |                          |
        v                          v
Safe event                  Needs review
        |                          |
        v                          v
Committed              Unmatched/Learning Queue
        |
        v
Patient Clinical History
```

Processing is asynchronous.

A successful import request normally means:

> The document has been accepted and queued for processing.

It does not mean extraction has already completed.

The caller should use the returned status URL to monitor progress.

Prerequisites

Before using the API:

1. Open Clinical History database migrations must be complete.
2. The API must be enabled in **Admin → Configuration**.
3. At least one API token must have been created.
4. The required LLM configuration must be available.
5. The SNOMED and image/SNOMED databases should have been built.
6. If the ingest queue is enabled, `queue_worker.php` must be running.

Relevant configuration settings include:

```
api_enabled
api_require_https
api_default_auto_commit
api_max_upload_mb

queue_enabled
queue_max_attempts
queue_job_timeout
queue_lease_seconds
```

Authentication

Every API request requires an Open Clinical History API token.

Tokens are created under:

Admin → Configuration → API Tokens

The preferred authentication method is an HTTP Bearer token:

```
Authorization: Bearer och_your_token_here
```

A fallback header is also supported:

```
X-API-Token: och_your_token_here
```

Bearer authentication should be preferred.

Token security

API tokens are secrets.

They should:

- never be embedded in publicly accessible client-side code
- never be committed to source control
- only be transmitted over HTTPS
- be stored using the secret-management facilities of the integrating system
- be revoked immediately if exposed

Open Clinical History stores only the SHA-256 hash of the token. The full plaintext token is displayed only once when it is created.

API Scopes

Tokens can carry three scopes.

Scope	Purpose
`import`	Submit patient history documents
`status`	Read document, processing and queue status
`commit`	Run the clinical audit/commit process

A normal integration using automatic commit generally requires:

```
import
status
```

```
commit
```

If `auto_commit` is enabled for an import, the calling token **must** have the `commit` scope as well as `import`.

A token without the required scope receives:

```
HTTP 403
```

with:

```
{
  "ok": false,
  "error": {
    "code": "insufficient_scope",
    "message": "This token does not carry the required scope."
  }
}
```

HTTPS

By default:

```
api_require_https = enabled
```

Requests made without HTTPS are rejected.

The API recognises HTTPS through:

- the PHP HTTPS server variable
- server port 443
- `X-Forwarded-Proto: https`

The latter allows Open Clinical History to operate correctly behind an HTTPS reverse proxy.

HTTPS should remain mandatory for environments containing clinical information.

Response Format

Every API response uses JSON.

Successful response:

```
{
  "ok": true,
  "data": {
  }
}
```

Error response:

```
{
  "ok": false,
  "error": {
    "code": "error_code",
    "message": "Description of the error."
  }
}
```

Applications should check both:

1. the HTTP status code; and
2. the `ok` property.

API Endpoints

The following actions are available.

Method	Action	Scope	Purpose
GET	`ping`	Valid token	Test authentication and API availability
POST	`import`	`import`	Submit a patient history
GET	`status`	`status`	Monitor extraction/import progress

Method	Action	Scope	Purpose
GET	`document`	`status`	Retrieve the document processing summary
POST	`commit`	`commit`	Audit and commit extracted events
GET	`queue`	`status`	Inspect the ingest queue
POST	`queue`	`status` + `import`	Retry or cancel queue work

It is recommended that callers always specify `action` explicitly.

Testing the Connection

Use:

```
GET history_api_import.php?action=ping
```

Example:

```
curl \
  -H "Authorization: Bearer och_your_token_here" \
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=ping"
```

A successful response resembles:

```
{
  "ok": true,
  "data": {
    "build": "2026-08-22.api-v2-app-config-llm",
    "token": {
      "id": 4,
      "label": "Clinical integration",
      "scopes": [
        "import",
        "status",
        "commit"
      ]
    },
    "date_format": "dd/mm/yyyy",
```

```
    "queue": {
      "enabled": true,
      "depth": {
        "queued": 0,
        "running": 1,
        "done": 42,
        "failed": 0,
        "cancelled": 0
      }
    },
    "server_time": "2026-08-25T19:45:00+10:00"
  }
}
```

This provides a useful initial connectivity test because it confirms:

- the API is enabled
- HTTPS is accepted
- the token is valid
- the token's available scopes
- the configured date format
- whether queue processing is enabled

Importing a Patient Record

Use:

```
POST history_api_import.php?action=import
```

The preferred request format is:

```
Content-Type: application/json
```

Example request:

```
{
  "record_number": "MRN-1001",
  "display_name": "Example Patient",
  "dob": "1967-04-02",
```

```
  "sex": "female",
  "source_name": "hospital-discharge-summary.txt",
  "text": "Patient clinical history goes here...",
  "auto_commit": true,
  "external_ref": "hospital-a:MRN-1001:discharge:84721"
}
```

Import Fields

`record_number`

Required

Example:

```
"record_number": "MRN-1001"
```

Maximum length:

```
64 characters
```

This is the primary external patient identifier used by the patient-import API.

Open Clinical History searches for an existing patient where:

```
patient_record_number = supplied record_number
source_system = history_import
```

If found, that patient is reused.

If not found, a new patient is created.

Important

The API is currently a **find-or-create patient API**, not a demographic-update API.

If the patient already exists, supplying different:

- display name
- DOB
- sex

does **not** update the existing patient record.

The `record_number` should therefore be stable for the lifetime of the patient within the source integration.

`display_name`

Optional

Example:

```
"display_name": "Example Patient"
```

Used as the patient's human-readable display name when a new patient is created.

It is stored with the patient's attributes.

It is not used as the primary patient-matching key.

`dob`

Optional

Preferred format:

```
YYYY-MM-DD
```

Example:

```
"dob": "1967-04-02"
```

ISO dates are recommended for all machine integrations.

The API also accepts the date format configured under:

Admin → Configuration → Date format

For example:

```
02/04/1967
```

when the system is configured for `dd/mm/yyyy`.

Validation

The date:

- must be valid
- cannot be in the future
- cannot have a year earlier than 1880

An invalid value produces:

```
bad_dob
```

Machine integrations should always use ISO `YYYY-MM-DD` to avoid regional date ambiguity.

---

sex

Optional

Accepted values are:

```
female
male
intersex
unknown
```

Values are case-insensitive.

If omitted:

```
unknown
```

is used.

An unrecognised value is currently normalised to:

```
unknown
```

rather than causing the request to fail.

---

source_name

Optional

Identifies the source document.

Example:

```
"source_name": "cardiology-letter-2026-08-25.txt"
```

Default:

```
api-upload.txt
```

A meaningful source name is strongly recommended because it makes individual documents easier to identify in Open Clinical History.

---

text

Required for JSON imports

Contains the complete unstructured source document.

Example:

```
"text": "12 March 2018 - Patient admitted with..."
```

The document can contain raw clinical text.

It does not need to be pre-classified into clinical events or SNOMED concepts.

Open Clinical History performs that processing.

The document must not be empty.

`auto_commit`

Optional

Boolean:

```
"auto_commit": true
```

or:

```
"auto_commit": false
```

If omitted, the value comes from:

```
api_default_auto_commit
```

in **Admin → Configuration**.

When `true`, the API token must have the:

```
commit
```

scope.

What auto-commit actually means

Auto-commit does **not** mean every extracted event is blindly inserted into the patient's clinical history.

After extraction, Open Clinical History performs its normal audit checks.

An event is automatically committed only when it satisfies the safety criteria.

Among other things, a proposal must:

- pass its clinical audit
- have a resolved SNOMED concept
- have no unresolved mapping warnings

- not represent a negated statement
- not represent family history
- not be merely a historical-summary statement

Safe events are committed.

Events that cannot be safely committed are rejected from automatic commit and routed to the unmatched/learning workflow for further resolution.

Conceptually:

```
Extracted proposal
        |
        v
Clinical audit
        |
        +---- safe -----------------> Commit
        |
        +---- uncertain/unsafe -----> Review/Learning Queue
```

This allows unattended imports without treating every AI-generated proposal as trusted clinical data.

`external_ref`

Optional but strongly recommended

Example:

```
"external_ref": "hospital-a:MRN-1001:discharge:84721"
```

This provides request-level idempotency.

If the same import request is retried with the same `external_ref`, Open Clinical History does not create another import.

Instead, it returns the existing document and its current processing state.

The response includes:

```
"duplicate": true
```

and returns HTTP:

```
200 OK
```

rather than creating a second queue item.

This is particularly important when an integration cannot determine whether an earlier HTTP request reached the server.

Recommended external reference design

Use a stable source-system identifier such as:

```
<system>:<patient>:<document-id>
```

For example:

```
hospital-a:MRN-1001:document-84721
```

or:

```
pms:48392:consultation-998312
```

The maximum stored length is approximately:

```
190 characters
```

The current implementation searches `external_ref` globally rather than per token, so integrations should ensure that external references are **globally unique across Open Clinical History**.

Optional `priority`

The JSON import endpoint also accepts:

```
"priority": 0
```

This controls ordering within the ingest queue.

Allowed effective values are:

```
-10 to +10
```

Higher numbers are processed before lower numbers.

For example:

```
"priority": 5
```

will be processed before a queued document with:

```
"priority": 0
```

Within the same priority, older queue items are processed first.

For normal integrations, leave this at:

```
0
```

unless there is a genuine requirement for priority processing.

Example JSON Import

```
curl -X POST \
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=import" \
  -H "Authorization: Bearer och_your_token_here" \
  -H "Content-Type: application/json" \
  -d '{
    "record_number": "MRN-1001",
    "display_name": "Example Patient",
    "dob": "1967-04-02",
    "sex": "female",
    "source_name": "specialist-letter.txt",
    "text": "Clinical document contents...",
    "auto_commit": true,
    "external_ref": "hospital-a:MRN-1001:letter-84721"
```

```
  }'
```

Successful Import Response

When queue processing is enabled, a successful import normally returns:

```
HTTP 202 Accepted
```

with a response similar to:

```
{
  "ok": true,
  "data": {
    "document_id": 142,
    "patient_id": 27,
    "queue_id": 81,
    "queue_position": 2,
    "state": "queued",
    "auto_commit": true,
    "external_ref": "hospital-a:MRN-1001:letter-84721",
    "status_url": "/demo/history_api_import.php?action=status&doc=142",
    "poll_after": 15
  }
}
```

`document_id`

Open Clinical History identifier for the uploaded source document.

Keep this identifier when monitoring the import.

`patient_id`

Internal Open Clinical History patient identifier.

This may refer to:

- an existing patient matched by `record_number`; or
- a newly created patient.

`queue_id`

Identifier of the ingest queue item.

`queue_position`

Number of queued items currently ahead of this item.

Therefore:

```
0
```

means no queued items are ahead of it.

Queue positions can change while workers process other records.

`state`

Immediately after queue submission:

```
queued
```

is normally returned.

`status_url`

Relative URL that can be used to monitor the document.

Example:

```
/demo/history_api_import.php?action=status&doc=142
```

Authentication is still required when using this URL.

`poll_after`

Suggested number of seconds before polling again.

In queue mode the API currently returns:

```
15
```

seconds.

Clients should respect this value rather than aggressively polling the server.

Importing a File

The API also supports:

```
multipart/form-data
```

Use a file part named:

```
history
```

For example:

```
curl -X POST \
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=import" \
  -H "Authorization: Bearer och_your_token_here" \
  -F "record_number=MRN-1001" \
  -F "display_name=Example Patient" \
  -F "dob=1967-04-02" \
  -F "sex=female" \
  -F "auto_commit=true" \
  -F "external_ref=hospital-a:MRN-1001:document-84721" \
```

```
  -F "history=@discharge-summary.txt"
```

When a file is uploaded, its filename becomes the document's `source_name`.

Maximum Document Size

The maximum accepted document size is controlled by:

```
api_max_upload_mb
```

under:

Admin → Configuration

Default:

```
2 MB
```

The limit applies to both:

- JSON `text`
- multipart `history` files

Documents exceeding the limit receive:

```
HTTP 413 Payload Too Large
```

with error code:

```
too_large
```

Character Encoding

Clinical text is expected to be UTF-8.

If uploaded text is not valid UTF-8, the current importer attempts to convert it from:

```
ISO-8859-1
```

to UTF-8 before storing it.

Integrating systems should supply UTF-8 directly wherever possible.

Duplicate Source Documents

Open Clinical History also calculates a SHA-256 hash of the normalised document text.

For a given patient, if identical document content has already been stored, the existing document can be reused rather than storing another physical copy.

However:

> `external_ref` should still be used as the integration's primary idempotency mechanism.

The document hash protects against identical content. `external_ref` identifies the source-system transaction/document itself.

Monitoring an Import

Use:

```
GET history_api_import.php?action=status&doc=<document_id>
```

Requires:

```
status
```

scope.

Example:

```
curl \
  -H "Authorization: Bearer och_your_token_here" \
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=status&doc=142"
```

Import States

The high-level API `state` may be:

State	Meaning
`queued`	Waiting for an ingest worker
`extracting`	Clinical extraction is running
`extracted`	Extraction finished; automatic commit was not requested
`committed`	Audit/commit processing has completed
`failed`	Processing failed after available attempts
`cancelled`	Queue work was manually cancelled

A typical automatic-import sequence is:

```
queued
   |
   v
extracting
   |
   v
committed
```

Without auto-commit:

```
queued
   |
   v
extracting
   |
   v
extracted
```

The client can then explicitly call the `commit` endpoint.

Example Status Response

A status response can resemble:


```
{
  "ok": true,
  "data": {
    "document_id": 142,
    "patient_id": 27,
    "source_name": "specialist-letter.txt",
    "document_state": "committed",
    "status_note": null,
    "char_count": 18492,
    "state": "committed",
    "queue": {
      "id": 81,
      "state": "done",
      "position": 0,
      "attempts": 1,
      "max_attempts": 3,
      "available_at": "2026-08-25 19:40:10",
      "started_at": "2026-08-25 19:40:12",
      "finished_at": "2026-08-25 19:41:44",
      "last_error": null
    },
    "auto_commit": true,
    "external_ref": "hospital-a:MRN-1001:letter-84721",
    "message": null,
    "job": "7a2d4c9e14f0b581",
    "extraction": {
      "state": "done",
      "phase": "complete",
      "chunks_total": 5,
      "chunks_done": 5,
      "events": 19,
      "requests": 7,
      "tokens": 14231,
```

```
      "started": 1787643612,
      "updated": 1787643704,
      "errors": []
    },
    "proposals": {
      "proposed": 0,
      "committed": 17,
      "rejected": 2
    },
    "committed": 17,
    "queued": 2,
    "status_url": "/demo/history_api_import.php?action=status&doc=142"
  }
}
```


The exact values depend on the document and current pipeline stage.

Understanding the Status Response

`document_state`

The current status stored against the source document itself.

This is distinct from the high-level API `state`.

For normal integrations, `state` should generally be used to decide what action to take next.

`char_count`

Number of characters in the stored source document.

`queue`

Contains ingest-queue information when queue mode is enabled.

`queue.state`

Possible queue-level states are:

```
queued
running
done
failed
cancelled
```

`attempts`

Number of times a worker has claimed the queue item.

A worker claim counts as an attempt even if that worker subsequently disappears.

`max_attempts`

Maximum number of processing attempts permitted before the item is permanently parked as failed.

`last_error`

The most recent queue processing error, if any.

A job that fails and is automatically retried may therefore have a `last_error` while returning to:

```
queued
```

Extraction Progress

The `extraction` object provides lower-level information about the extraction worker.

It may contain:

```
state
phase
chunks_total
chunks_done
events
requests
tokens
started
updated
errors
```

This can be used to provide richer progress information to an integrating application.

The `extraction` object may be:

```
null
```

before a worker job has been created.

Proposal Counts

The status response contains:

```
"proposals": {
  "proposed": 0,
  "committed": 17,
  "rejected": 2
}
```

These are the extracted clinical event proposals associated with the document.

`proposed`

Events that still await a final decision.

`committed`

Events accepted into the patient's clinical history.

`rejected`

Events not automatically committed.

Rejected does not necessarily mean that the extracted clinical statement was incorrect.

It may mean that Open Clinical History could not safely establish:

- an appropriate SNOMED concept
- sufficient specificity
- an appropriate anatomical mapping
- a clean audit result

Such events can be routed through the unmatched/learning process.

A Note About the Top-Level `queued` Value

There are two different queue concepts in the API response.

`queue`

The object:

```
"queue": {
  "state": "running"
}
```

describes the **document ingest queue**.

`queued`

The top-level numeric value:

```
"queued": 2
```

after the audit stage means that two extracted clinical events were routed to the **unmatched/learning workflow** rather than automatically committed.

It does **not** mean there are two documents ahead of this document in the ingest queue.

Status by Job Identifier

Status can also be requested using the extraction job identifier:

```
GET history_api_import.php?action=status&job=<job>
```

Example:

```
curl \
  -H "Authorization: Bearer och_your_token_here" \

"https://www.openclinicalhistory.org/demo/history_api_import.php?action=status&job=7a2d4c9e14f0b581"
```

Job identifiers are 16 lowercase hexadecimal characters.

For most integrations, monitoring by:

```
document_id
```

is simpler.

Document Endpoint

Use:

```
GET history_api_import.php?action=document&doc=<document_id>
```

Requires:

```
status
```

scope.

Example:

```
curl \
  -H "Authorization: Bearer och_your_token_here" \
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=document&doc=142"
```

The current implementation returns the same assembled document-processing view used by document-based status requests.

This endpoint provides a semantic way for an integration to request the current summary for a known document.

Manual Commit

If the document was imported with:

```
"auto_commit": false
```

the normal completed state is:

```
extracted
```

To run the audit and commit stage:

```
POST history_api_import.php?action=commit&doc=<document_id>
```

Requires:

```
commit
```

scope.

Example:

```
curl -X POST \
  -H "Authorization: Bearer och_your_token_here" \
```

```
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=commit&doc=142"
```

A successful response resembles:

```
{
  "ok": true,
  "data": {
    "document_id": 142,
    "patient_id": 27,
    "state": "committed",
    "committed": 17,
    "queued": 2,
    "errors": []
  }
}
```

Again:

```
committed
```

is the number of clinical events accepted into the clinical history.

```
queued
```

is the number routed to the unmatched/learning workflow.

Commit While Extraction Is Running

The API will not allow a document to be committed while extraction is still active.

It returns:

```
HTTP 409 Conflict
```

with:

```
still_extracting
```

The response includes a `status_url` that should be polled until extraction has finished.

Ingest Queue

When:

```
queue_enabled = true
```

API imports are placed into the controlled ingest queue.

This prevents a burst of API submissions from launching an unlimited number of LLM extraction processes.

The queue worker processes them in a controlled sequence.

Inspecting the Queue

Use:

```
GET history_api_import.php?action=queue
```

Requires:

```
status
```

scope.

Example:

```
curl \
  -H "Authorization: Bearer och_your_token_here" \
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=queue"
```

The response includes:

- queue depth by state
- recent queue items

By default the most recent:

```
25
```

items are returned.

A custom limit can be supplied:

```
?action=queue&limit=50
```

The API permits between:

```
1 and 100
```

items through this endpoint.

Inspecting One Queue Item

Use:

```
GET history_api_import.php?action=queue&id=<queue_id>
```

For example:

```
curl \
  -H "Authorization: Bearer och_your_token_here" \
  "https://www.openclinicalhistory.org/demo/history_api_import.php?action=queue&id=81"
```

The response includes the queue item and its current position.

Retry a Failed Queue Item

A failed or cancelled item can be reset and submitted again.

Use:

```
POST history_api_import.php?action=queue&id=<queue_id>&op=retry
```

Requires both:

```
status
import
```

scopes.

Retry:

- changes the item back to `queued`
- resets attempts to zero
- makes the item immediately available
- clears the previous worker lease
- clears the previous queue error

Cancel a Queue Item

Use:

```
POST history_api_import.php?action=queue&id=<queue_id>&op=cancel
```

Requires:

```
status
import
```

scopes.

Cancellation is allowed when the queue item is:

```
queued
```

or:

```
failed
```

A currently running worker is not cancelled by this endpoint.

Automatic Queue Retries

If processing fails, the queue automatically retries until:

```
queue_max_attempts
```

has been reached.

Retry delay uses increasing quadratic backoff.

Conceptually, with successive failed attempts:

```
attempt 1 -> wait 60 seconds
attempt 2 -> wait 240 seconds
attempt 3 -> wait 540 seconds
```

The delay is capped at one hour.

Once all allowed attempts have been exhausted, the item is marked:

```
failed
```

Worker Leases

When a worker claims a queue item, it receives a temporary lease.

The lease is controlled by:

```
queue_lease_seconds
```

Workers extend their lease while long-running extraction is taking place.

If a worker crashes and its lease expires, the item can be returned to the queue rather than remaining permanently stuck in:

```
running
```

Recommended Client Workflow

A robust integration should use the following pattern:

```
1. Generate stable external_ref
        |
        v
2. POST action=import
        |
        +---- HTTP 202 ----+
        |                  |
        v                  |
   Save document_id        |
   and queue_id            |
        |                  |
        v                  |
3. Wait poll_after         |
        |                  |
        v                  |
4. GET action=status       |
        |                  |
        +---- queued ------+
        |
        +---- extracting --+
        |
        +---- committed ----------> Complete
        |
        +---- extracted ----------> POST commit if required
        |
        +---- failed -------------> Investigate/retry
        |
        +---- cancelled ----------> Stop/re-submit if appropriate
```

Do not continuously resubmit the original document while waiting.

Poll the status endpoint instead.

If network uncertainty makes a resubmission necessary, send the **same** `external_ref`.

Recommended Polling Behaviour

After submission, use the returned:

```
"poll_after": 15
```

value.

A simple client strategy is:

```
Submit
  |
wait 15 seconds
  |
check status
  |
still processing?
  |
yes -> wait again
  |
no -> process result
```

There is no need to poll every second.

Clinical extraction may involve several model requests and can take some time for large documents.

Idempotent Retry Example

Initial request:

```
{
  "record_number": "MRN-1001",
  "text": "Clinical history...",
  "external_ref": "hospital-a:document-84721",
  "auto_commit": true
}
```

Suppose the client's connection fails before it receives the response.

The client may safely resend:

```
{
  "record_number": "MRN-1001",
  "text": "Clinical history...",
  "external_ref": "hospital-a:document-84721",
  "auto_commit": true
}
```

If the original submission was already accepted, the API returns the existing request with:

```
"duplicate": true
```

rather than intentionally creating another import.

Patient Matching Behaviour

Patient matching deserves particular attention when designing an integration.

Open Clinical History currently matches imported patients using:

```
patient_record_number
+
source_system = history_import
```

It does **not** currently match using:

- display name
- DOB
- sex
- document source

- external reference

Therefore:

```
MRN-1001
```

must consistently refer to the same patient.

Existing patient

If the patient already exists:

```
reuse patient
```

New record number

If it does not exist:

```
create new patient
```

Consequently, changing a patient's external record number can create another Open Clinical History patient rather than updating the existing one.

Patient Asset Set

When a patient is first created by this API:

```
sex = male
```

selects the:

```
male
```

anatomical asset set.

The current implementation uses the:

```
female
```

asset set for the other sex values.

This affects the anatomical visual layers used when displaying the patient's history.

Date Handling Recommendation

Although Open Clinical History can accept the configured display format, integrations should always send:

```
YYYY-MM-DD
```

For example:

```
"dob": "1971-09-05"
```

rather than:

```
"dob": "05/09/1971"
```

This makes the payload independent of the user-interface date configuration and avoids ambiguity between Australian and US date formats.

HTTP Status Codes

Common HTTP responses include:

HTTP	Meaning
`200`	Request completed successfully
`202`	Import accepted for asynchronous processing
`400`	Invalid request or field

HTTP	Meaning
`401`	Missing, invalid, expired or revoked token
`403`	Token does not have the required scope
`404`	Document, job or queue item does not exist
`405`	Incorrect HTTP method
`409`	Current state does not permit the requested operation
`413`	Document exceeds the configured size limit
`422`	Document/patient could not be stored or linked
`500`	Internal application error
`503`	API, queue or worker unavailable/configuration incomplete

---

Common API Error Codes

Authentication

```
missing_token
invalid_token
insufficient_scope
```

Configuration

```
not_configured
api_disabled
https_required
queue_disabled
```

Request format

```
bad_json
method_not_allowed
unknown_action
missing_target
```

Document import

```
missing_text
empty_document
too_large
upload_failed
missing_record_number
bad_record_number
bad_dob
store_failed
link_failed
worker_unavailable
```

Processing

```
still_extracting
bad_job
not_found
```

Queue management

```
unknown_op
not_applicable
```

Example Error

An invalid date may return:

```
HTTP 400
```

```
{
  "ok": false,
  "error": {
    "code": "bad_dob",
    "message": "dob must be ISO YYYY-MM-DD or the configured display format (dd/mm/yyyy)."
  }
}
```

Applications should use the machine-readable:

```
error.code
```

for program logic and treat:

```
error.message
```

as diagnostic information.

Queue Worker Requirement

When the ingest queue is enabled:

```
queue_enabled = true
```

submitting a document does **not** directly perform extraction.

It creates a queue item.

At least one instance of:

```
queue_worker.php
```

must therefore be running.

If no worker is running, submissions can still return:

```
202 Accepted
```

but will remain in:

```
queued
```

until a worker becomes available.

Queue health should therefore be monitored as part of production operations.

Behaviour Without the Queue

If the `ingest_queue` table is unavailable or:

```
queue_enabled = false
```

the API falls back to its earlier behaviour and spawns an extraction worker for each import.

The response still returns:

```
202 Accepted
```

but contains a job identifier rather than queue information.

The suggested poll interval becomes:

```
5 seconds
```

This mode is unbounded and is less appropriate for burst or high-volume integrations.

The controlled ingest queue is recommended.

API Security Model

API permissions are capability based.

A token with:

```
status
```

scope can currently query API document and queue status generally.

A token with:

```
commit
```

scope can request a commit for a supplied document identifier.

The current API does not restrict documents to the token that originally submitted them.

Therefore:

> API tokens should currently be treated as trusted integration credentials for the Open Clinical History installation, rather than as isolated per-patient or per-tenant credentials.

Use least-privilege scopes and issue separate tokens for different integrations where appropriate.

Suggested Integration Requirements

A production integration should:

- use HTTPS
- store API tokens securely
- use ISO dates
- use a stable patient `record_number`
- provide a meaningful `source_name`
- provide a globally unique `external_ref`
- store the returned `document_id`
- respect `poll_after`
- poll status rather than repeatedly submitting

- handle asynchronous processing
- handle failed imports explicitly
- use API error codes rather than parsing error messages
- avoid assuming every extracted event will be automatically committed
- monitor ingest-queue health

Minimum JSON Request

The smallest valid JSON import is:

```
{
  "record_number": "MRN-1001",
  "text": "Clinical history text"
}
```

All other fields have defaults or are optional.

For a real integration, however, the recommended payload is:

```
{
  "record_number": "MRN-1001",
  "display_name": "Example Patient",
  "dob": "1967-04-02",
  "sex": "female",
  "source_name": "specialist-letter-84721.txt",
  "text": "Clinical history text...",
  "auto_commit": true,
  "external_ref": "source-system:MRN-1001:84721"
}
```

Recommended Production Sequence

```
Create API token
        |
        v
GET ping
        |
        v
POST import
        |
        v
Store document_id
        |
        v
Poll status
        |
        +---- queued/extracting ---> continue polling
        |
        +---- committed -----------> finished
        |
        +---- extracted -----------> commit if required
        |
        +---- failed --------------> investigate/retry
```

Summary

`history_api_import.php` provides the machine-to-machine entry point into the Open Clinical History patient-processing pipeline.

The API deliberately separates:

```
Submission
    |
    v
Asynchronous extraction
    |
    v
Clinical audit
    |
```

```
    v
Safe commit
```

This means external systems can provide **raw, unstructured clinical records** while Open Clinical History performs the extraction, SNOMED classification, anatomical mapping and clinical-safety checks required to turn those records into a longitudinal patient history.

The external system does not need to pre-classify the clinical content before submission.