SNOMED Re-index / Learning Queue

The **SNOMED Re-index / Learning Queue** manages clinical terms that Open Clinical History could not safely resolve during patient-history import.

The page is:

```
/process_queue.php
```

During patient import, Open Clinical History deliberately avoids guessing when it cannot establish a sufficiently reliable terminology or anatomical mapping.

Instead, unresolved items are placed into:

```
history_unmatched_queue
```

The Learning Queue allows those items to be reconsidered later using the latest:

- local SNOMED CT dictionaries
- SNOMED terminology indexes
- curated condition catalogue
- curated aliases
- terminology matching logic
- configured grounded LLM resolution

This allows terminology coverage to improve over time without re-extracting the original patient document.

Why the Learning Queue Exists

Clinical language is highly variable.

The same clinical concept may appear in source records using:

- formal terminology
- abbreviations
- local terminology
- spelling variations
- historical terminology
- clinician shorthand
- descriptive phrases
- overly specific or ambiguous wording

For example:

```
myocardial infarction

heart attack

acute MI

NSTEMI

non-ST elevation myocardial infarction
```

may refer to related clinical concepts, but Open Clinical History should not simply assume that every unfamiliar expression means the same thing.

The import pipeline therefore follows a conservative rule:

> If a clinical event cannot be resolved safely, retain it for later learning rather than inventing a mapping.

Conceptually:

```
Patient document
      |
      v
Clinical event extracted
      |
      v
SNOMED resolution
      |
```

```
      +---- reliable match ------> continue import
      |
      +---- unresolved/unsafe
                    |
                    v
         history_unmatched_queue
                    |
                    v
              Learning Queue
```

This Is Not LLM Training

The term **Learning Queue** does not mean that Open Clinical History trains Gemini or modifies an underlying AI model.

No model weights are changed.

Instead, learning occurs through the application's own controlled terminology layer.

For example:

```
Previously unknown clinical term
          |
          v
Add/refine local terminology or alias
          |
          v
Re-run terminology resolution
          |
          v
Valid local SNOMED concept found
          |
          v
Update original event proposal
```

The Open Clinical History terminology database becomes more capable of recognising the language encountered in real clinical records.

Two-Stage Learning

The current architecture deliberately separates terminology learning from changing the patient record.

```
STAGE 1
Terminology learning / re-indexing

process_queue.php
        |
        v
Can this previously unresolved term
now be mapped to SNOMED?
        |
        v
Update event_proposal
        |
        v
Mark learning item resolved



STAGE 2
Apply learned mapping

queue_worker.php
        |
        v
Re-evaluate anatomical mapping
        |
        v
Reapply clinical safety gates
        |
        +---- safe ------> clinical history
        |
        +---- unsafe ----> held for review
```

`process_queue.php` performs **Stage 1**.

Resolving an item on this page does **not by itself add the event to the patient's clinical history**.

That distinction is important.

How Items Enter the Learning Queue

During Audit & Import, an extracted event is automatically committed only when it passes the required safety conditions.

Broadly:

```
Audit passed
AND
SNOMED concept resolved
AND
not negated
AND
not family history
AND
not merely historical-summary text
AND
no anatomical mapping warnings
```

If an event cannot safely be committed, its proposal is rejected from automatic import and an entry is written to:

```
history_unmatched_queue
```

The original event proposal is preserved.

Learning Queue Item Types

Each queue item has an `unmatched_type`.

The current values are:

```
snomed_concept
body_layer
specificity
```

SNOMED Concept

```
snomed_concept
```

means that no suitable SNOMED concept was available when automatic import was attempted.

Conceptually:

```
Clinical event
      |
      v
No sufficiently reliable SNOMED match
      |
      v
SNOMED_CONCEPT learning item
```

This is the most direct terminology-learning case.

Body Layer

```
body_layer
```

means a SNOMED concept was available, but the anatomical-image mapping contained warnings.

For example:

```
Clinical concept resolved
        |
```

```
        v
Anatomical mapping ambiguous
        |
        v
BODY_LAYER learning item
```

A later learning/application cycle can re-evaluate the event using the current image/SNOMED database.

Specificity

```
specificity
```

is used where a SNOMED concept exists and there are no body-layer warnings, but another automatic-import safety condition prevented commitment.

It represents a broader class of events requiring further resolution or review.

Opening the Learning Queue

Open:

Process Learning Queue

from the administration menu.

The page first displays the current state of terminology learning.

Queue Statistics

The statistics at the top of the page provide several different views of the learning backlog.

Need Re-indexing

Example:

```
73
NEED RE-INDEXING
8 document(s)
```

This is the number of rows in:

```
history_unmatched_queue
```

whose:

```
review_status = pending
```

These items have not yet been successfully resolved.

The document count shows how many different source documents contain those pending items.

Successfully Re-indexed

Example:

```
568
SUCCESSFULLY RE-INDEXED
20 document(s)
```

These are learning rows where:

```
review_status = resolved
```

A later terminology-resolution attempt successfully found a SNOMED concept.

When this occurs, the linked:

```
event_proposal
```

is updated with the new terminology match.

This does **not necessarily mean the event has already been applied to the patient timeline**.

Terminology resolution and patient-history application are separate stages.

Still Unmatched

Example:

```
7
STILL UNMATCHED
Already attempted, still pending
```

These are pending learning items that have already been through at least one re-indexing attempt but still could not be resolved.

Internally these are identified by a resolution note beginning with:

```
Still unmatched
```

The current note written by the processor is:

```
Still unmatched. Add to condition_alias table.
```

These items remain in the queue so they can be tried again after terminology resources have improved.

Awaiting First Attempt

Example:

```
66
AWAITING FIRST ATTEMPT
Pending with no processing result yet
```

These are queue items where:

```
review_status = pending
```

and there is no existing processing result in `resolution_notes`.

They have not yet been processed by the Learning Queue.

Ignored

Example:

```
0
IGNORED
Blank / deliberately skipped queue items
```

These are queue rows where:

```
review_status = ignored
```

The current processor automatically ignores an item when there is no usable search term.

For example:

```
condition_text = blank

and

event_title = blank
```

produces:

```
Skipped: Blank search term
```

and the queue item is marked ignored.

Unresolved Proposals

The **Unresolved Proposals** statistic examines the underlying:

```
event_proposal
```

table.

It counts proposals in the relevant active states where:

```
matched_concept_id IS NULL
```

This is a different measurement from the Learning Queue itself.

The Learning Queue measures:

```
history_unmatched_queue
```

while this metric looks directly at:

```
event_proposal
```

It is therefore possible for these figures to differ.

Unresolved Not in Pending Queue

This statistic highlights a possible difference between unresolved event proposals and pending learning rows.

The current calculation is:

```
unresolved proposals - pending learning items
```

with the result prevented from falling below zero.

It is intended as a diagnostic indicator rather than an exact row-by-row reconciliation.

A non-zero value can indicate unresolved proposal work that has not yet appeared as pending terminology-learning work.

Queue Items Ever

Example:

```
641
QUEUE ITEMS EVER
568 processed
```

This is the total historical number of rows in:

```
history_unmatched_queue
```

including:

- pending
- resolved
- ignored

The processed value is:

```
resolved + ignored
```

Learning Queue Completion

The completion bar shows how much of the historical Learning Queue has reached a terminal learning state.

The calculation is:

```
resolved + ignored
------------------ × 100
 total queue rows
```

For example:

```
568 resolved
0 ignored
73 pending
```

```
641 total
```

produces:

```
88.6% processed
```

Resolution Rate

The page also displays a **resolution rate**.

This is intended to answer:

> Of the items that have actually received a meaningful resolution attempt, how many have been successfully matched?

The current calculation uses:

```
resolved
---------------------------------- × 100
resolved + still unmatched + ignored
```

New items awaiting their first attempt are excluded.

For example:

```
568 resolved
7 still unmatched
0 ignored
```

produces:

```
98.8%
```

Manual Queue Processing

The main action on the page is:

Process Next 50 Items

The Learning Queue is deliberately processed in bounded batches.

Each request processes up to:

```
50
```

pending items.

This prevents one browser request from attempting to resolve the entire learning database in a single operation.

Which Items Are Processed First?

Pending records are selected using:

```
updated_at ASC
id ASC
```

In other words, the oldest pending work is selected first.

Up to 50 rows are loaded.

Queue Rotation After an Unsuccessful Attempt

When an item remains unmatched, its:

```
updated_at
```

timestamp is updated.

Because pending work is ordered by the oldest `updated_at`, the unsuccessful item effectively moves towards the back of the queue.

For example:

```
73 pending


Process first 50
       |
       +---- 45 resolved
       |
       +---- 5 still unmatched
                     |
                     v
              updated_at = now
                     |
                     v
             move behind older
             unattempted items
```

This allows new/unattempted work to receive a first attempt before repeatedly retrying the same difficult terms.

Preventing Concurrent Processing

The selected rows are loaded inside a database transaction using:

```
FOR UPDATE
```

This locks the batch while it is being processed.

The purpose is to prevent two browser requests from processing the same Learning Queue rows simultaneously.

Conceptually:

```
Administrator A
      |
      v
locks next 50 rows
      |
      +-------------------------+
                                |
Administrator B                 |
      |                         |
      v                         |
cannot independently modify     |
those same locked rows          |
                                |
                        transaction completes
```

CSRF Protection

The processing action also requires a session-specific CSRF token.

This prevents another website from causing an authenticated administrator's browser to submit the Learning Queue processing form without their intention.

Choosing the Search Term

For each queue item, Open Clinical History uses:

```
condition_text
```

if available.

Otherwise it falls back to:

```
event_title
```

Conceptually:

```
condition_text present?
      |
   yes|       no
      |        |
      v        v
condition   event_title
   text
      \        /
       \      /
        v    v
       search term
```

If both are empty, the item is ignored.

How Re-indexing Works

The queue does more than perform a simple text lookup.

For each term it calls the same terminology-resolution service used by the clinical ingestion pipeline:

```
HistoryIngest::matchCondition()
```

This allows the item to benefit from improvements made since the original document was processed.

---

1. Normalise the Clinical Term

The search text is first normalised.

Open Clinical History also creates useful clinical variations where appropriate.

Examples include UK/US spelling variants such as:

```
ischaemia   ↔ ischemia
oedema      ↔ edema
haemorrhage ↔ hemorrhage
anaemia     ↔ anemia
tumour      ↔ tumor
apnoea      ↔ apnea
```

and terminology variants such as:

```
appendicectomy ↔ appendectomy
```

---

2. Simplify Clinical Phrases

The matcher can generate less contextual variants of a term.

For example:

```
history of myocardial infarction
```

may also be searched as:

```
myocardial infarction
```

Likewise modifiers such as:

```
previous
prior
diagnosis of
diagnosed with
residual
```

can be removed for candidate discovery.

Some severity or temporal prefixes may also be stripped when searching for the underlying clinical concept.

These transformations help retrieve candidates.

They do not by themselves automatically determine the final SNOMED concept.

---

3. Check the Curated Condition Catalogue and Aliases

If installed, Open Clinical History checks:

```
condition_catalogue
condition_alias
```

first.

An alias provides a controlled mapping between terminology encountered in source records and a known catalogue entry.

Conceptually:

```
Local clinical expression
        |
        v
condition_alias
        |
        v
condition_catalogue
        |
        v
SNOMED concept
```

This is one of the principal mechanisms by which the system can learn terminology encountered in real-world clinical data.

Why Aliases Matter

Suppose repeated source records use:

```
local clinical expression X
```

and Open Clinical History cannot reliably resolve it.

After an administrator establishes that it should map to a known catalogue concept, an appropriate alias can be added.

The next Learning Queue pass can then resolve:

```
local clinical expression X
        |
        v
alias recognised
        |
        v
known catalogue concept
        |
        v
SNOMED CT
```

The source patient record does not need to be extracted again.

---

4. Search the Local SNOMED Dictionary

If no curated match exists, the matcher searches the compact SNOMED indexes:

```
snomed_health_history_lookup
snomed_health_history_term_lookup
```

It first looks for exact matches.

A single semantically compatible exact result can be accepted deterministically.

Possible exact methods include:

```
exact_fsn
exact_synonym
```

---

5. Search for SNOMED Candidates

When a unique exact result is unavailable, Open Clinical History searches its local SNOMED terminology index for candidate concepts.

Potential candidates are ranked locally before any AI involvement.

The number of candidates is bounded by the configured terminology-resolution limits.

---

6. Grounded AI Resolution

If an LLM is configured, ambiguous candidate sets can be passed through the grounded SNOMED-resolution process.

Importantly, the LLM does **not** supply an arbitrary SNOMED ID.

Instead:

```
Clinical term
      |
      v
Local SNOMED search
      |
      v
Candidate A
Candidate B
Candidate C
      |
      v
LLM selects from supplied candidates
```

```
      |
      v
Local SNOMED concept
```

The LLM receives opaque candidate keys and must select one of the candidates Open Clinical History supplied.

It is explicitly instructed not to invent a SNOMED identifier.

Grounded Repair

Where enabled, the normal SNOMED repair process may also be used.

This can generate revised terminology search expressions and retry local candidate resolution.

The number of repair rounds and candidate limits are controlled under:

Admin → Configuration

using the grounded SNOMED settings.

Fallback if the LLM Fails

If AI-assisted resolution raises an error, `process_queue.php` does not automatically abandon the queue batch.

`matchCondition()` falls back to deterministic local candidate matching.

This allows local terminology improvements and exact aliases to remain useful even when the configured LLM is temporarily unavailable.

When a Match Is Found

If a valid SNOMED concept is found, the linked:

```
event_proposal
```

is updated.

The following values are written:

```
matched_concept_id
matched_term
match_method
match_confidence
```

For example, conceptually:

```
event_proposal #482

Before:
matched_concept_id = NULL

After:
matched_concept_id = <SNOMED ID>
matched_term       = <SNOMED term>
match_method       = <resolution method>
match_confidence   = <confidence>
```

The Learning Queue row is then changed to:

```
review_status = resolved
```

with a resolution note similar to:

```
Matched to SNOMED ID: <concept-id> (<term>)
```

Successfully Re-indexed Does Not Mean Committed

This is one of the most important behaviours of the page.

When the queue says:

```
Successfully re-indexed
```

it means:

> A previously unresolved event proposal now has a SNOMED terminology match.

It does **not** mean:

> The event has automatically been inserted into the patient's clinical history.

The original clinical safety controls are deliberately retained.

Stage 2: Applying Learned Mappings

The separate learned-mapping application process can subsequently reconsider the event.

It uses the newly resolved SNOMED concept to:

1. recalculate anatomical layers
2. update the event proposal mapping information
3. reapply the original audit/safety conditions
4. either commit the event or hold it for review

Conceptually:

```
process_queue.php
        |
        v
SNOMED learned
        |
        v
event_proposal updated
```

```
        |
        v
Apply Learned Mapping
        |
        v
Rebuild anatomy
        |
        v
Safety gate
        |
        +---- safe ----------> Commit
        |
        +---- not safe ------> Hold
```

The original source-document extraction is not repeated.

Existing Committed Events

The two-stage learning architecture also supports updating an event that has already been committed.

When learned terminology is later applied to such an event, Open Clinical History can refresh:

- SNOMED coding
- anatomical mapping
- associated visual sites

without creating another clinical event.

The operation records that the coding was updated through the learning/re-index process.

Learned Mapping Safety Gates

A newly learned terminology match is still not sufficient by itself for automatic commitment.

The application stage checks conditions including:

- the original clinical audit passed
- the source was not negated
- the source was not family history
- the source was not merely a historical summary
- the terminology match is sufficiently strong
- the matching method is sufficiently reliable
- required anatomical mappings are available

The current learned-application code requires a strong terminology result for unattended commitment.

Weak results remain held.

Still-Unmatched Items

If no suitable SNOMED concept is found, the queue item remains:

```
review_status = pending
```

and receives:

```
Still unmatched. Add to condition_alias table.
```

Its `updated_at` timestamp is refreshed so it moves towards the back of the pending queue.

The item can then be reconsidered after changes such as:

- adding a curated alias
- correcting a catalogue entry
- rebuilding the SNOMED indexes
- installing a newer SNOMED release
- improving terminology matching
- improving grounded resolution

Processing a New Batch

Click:

Process Next 50 Items

again whenever terminology resources have changed or more queue work remains.

The button automatically shows the number of items in the next batch.

For example:

```
Process Next 50 Items (50)
```

or, where fewer remain:

```
Process Next 50 Items (23)
```

When there are no pending items, the button is disabled.

Processing Result

After a batch completes, the page reports:

```
Queue processing complete.
Evaluated 50 term(s):
42 re-indexed,
8 still unmatched,
0 ignored.
```

Note that blank items marked ignored are not included in the `Evaluated` count because no terminology lookup was attempted for them.

Outstanding: Next in Queue

The **Outstanding** table displays up to the next 50 pending records.

Columns include:

Column	Meaning
ID	Learning Queue row identifier

Column	Meaning
Type	`SNOMED_CONCEPT`, `BODY_LAYER` or `SPECIFICITY`
Original Term	Clinical term being reconsidered
Attempt State	Whether this is new or has previously failed
Last Attempt Notes	Result of the previous attempt

Attempt State: NEW

```
NEW
```

means no previous Learning Queue result has been recorded.

The notes display:

```
Awaiting first processing attempt...
```

Attempt State: RETRY NEEDED

```
RETRY NEEDED
```

means the term has already been processed but could not yet be resolved.

The notes display:

```
Still unmatched. Add to condition_alias table.
```

Because failed attempts are moved towards the back of the queue, newly encountered terms are given an opportunity before difficult terms are repeatedly retried.

Processed Log

The bottom of the page displays the most recent:

```
50
```

non-pending Learning Queue items.

These include:

```
resolved
ignored
```

The table shows:

- queue ID
- original clinical term
- final learning status
- resolution notes

For a resolved item, the notes contain the matched SNOMED concept.

For example:

```
Matched to SNOMED ID: <concept-id> (<SNOMED term>)
```

When Should the Learning Queue Be Processed?

The queue is particularly useful after:

- patient histories have produced unresolved terminology
- curated aliases have been added
- terminology dictionaries have been changed
- SNOMED CT has been updated
- SNOMED indexes have been rebuilt
- terminology-resolution logic has been improved

- anatomical mappings have been rebuilt

A typical learning cycle is:

```
Patient imports
      |
      v
Unresolved terminology accumulates
      |
      v
Review recurring terminology gaps
      |
      v
Improve aliases/catalogue/SNOMED data
      |
      v
Process Learning Queue
      |
      v
Previously unresolved terms resolve
      |
      v
Apply learned mappings
      |
      v
Patient histories improve
```

Relationship to SNOMED Updates

Installing a newer SNOMED CT release may introduce:

- new concepts
- new descriptions
- changed preferred terminology
- additional synonyms

After importing the new release and rebuilding the required lookup databases, the Learning Queue can be processed again.

A term that previously had no valid local match may now resolve.

This avoids having to re-import the patient's original source document merely because the terminology database improved.

Relationship to the Image/SNOMED Database

A terminology resolution can also change the anatomical mapping available to an event.

After a SNOMED concept has been learned, the second-stage application process uses the current:

```
body_layer_lookup
body_layer_term
```

tables to determine the appropriate visual layers.

This means improvements made through:

Image/SNOMED Summary Build

can also benefit previously unresolved patient events.

Important Distinction Between the Queues

Open Clinical History contains more than one queue with different responsibilities.

Ingest Queue

```
ingest_queue
```

controls document-processing jobs.

It manages:

- patient-document extraction
- worker concurrency
- retries
- worker leases
- learned-mapping application jobs

Learning Queue

```
history_unmatched_queue
```

contains unresolved **clinical event mappings**.

It manages terminology and mapping gaps discovered during patient processing.

Conceptually:

```
INGEST QUEUE

document
   |
   v
worker
   |
   v
clinical extraction



LEARNING QUEUE

unresolved clinical event
   |
   v
terminology re-index
   |
```

```
   v
learned mapping
```

They should not be confused.

Recommended Operational Workflow

When unresolved terminology is present:

```
1. Open Process Learning Queue
        |
        v
2. Review outstanding terms
        |
        v
3. Identify recurring terminology gaps
        |
        v
4. Improve curated aliases/catalogue
   or SNOMED/index data where appropriate
        |
        v
5. Process Next 50 Items
        |
        v
6. Review successful and unmatched counts
        |
        +---- unresolved ----> investigate/curate
        |
        +---- resolved ------> learned mapping ready
        |
        v
7. Apply learned mappings
        |
```

```
        v
8. Review held items
```

Do Not Add Aliases Blindly

A recurring unfamiliar term should not automatically be assigned to the first plausible SNOMED concept.

Before adding a curated alias, confirm that:

- the source expression has a stable clinical meaning
- the chosen SNOMED concept represents that meaning
- the semantic type is appropriate
- the mapping is not more specific than the source
- laterality is not being incorrectly encoded
- an abbreviation is not ambiguous between specialties

For example, an alias should not convert an ambiguous abbreviation into a specific diagnosis unless that meaning is genuinely reliable within the intended context.

The learning mechanism is deliberately governed because the resulting mappings can eventually become part of a patient's clinical record.

Summary

`process_queue.php` is the **terminology-learning and SNOMED re-indexing stage** of Open Clinical History.

It takes events that were previously unresolved:

```
Unresolved clinical term
```

and asks:

```
Can the current Open Clinical History
terminology system resolve this now?
```

using:

```
Curated aliases
      +
Local SNOMED CT
      +
Clinical spelling/term normalisation
      +
Local candidate search
      +
Bounded grounded AI resolution
```

When a reliable concept is found:

```
history_unmatched_queue
        |
        v
resolved

event_proposal
        |
        v
SNOMED mapping updated
```

A separate governed application stage then determines whether that learned mapping can safely update the patient's clinical history.

This separation allows Open Clinical History to **learn from unresolved terminology without bypassing the clinical safety controls that govern the patient record**.

Revision #1
Created 2026-08-25 10:08:21 UTC by Admin
Updated 2026-08-25 10:08:47 UTC by Admin