**SDMX Glossary Generation Project Description**

# General Overview

The project is designed to replace the outdated SDMX glossary model (located at [http://purl.org/linked-data/sdmx/2009/concept](http://purl.org/linked-data/sdmx/2009/concept?utm_source=chatgpt.com)) with a new one, hosted at <http://purl.semanticip.org/linked-data/sdmx/concept>.The main goal of the project is the automated transformation of concepts from an XML document into an RDF model in Turtle (TTL) format, taking into account adjustments and tuning of correspondences between the new and the outdated models. The developed software is a tool for generating glossaries in RDF Data Cube format, converting concepts from XML and the current Turtle model into the new RDF model.

# Project Architecture

The software consists of the following core modules:

## 2.1 Main script — sdmxglossgen.py

This module controls the transformation process (using other functions):

* Loads the XML document with source concepts and the RDF graph of the outdated model.
* Defines namespaces for data processing.
* Extracts concepts from the XML document.
* Builds the new RDF model:
  + Creates Turtle prefixes.
  + Defines the conceptual scheme (skos:ConceptScheme).
  + Processes relations between concepts (skos:broader, skos:narrower, skos:related, skos:exactMatch).
* Writes the RDF model in Turtle format.
* Creates a CSV file with code list associations.
* Performs tuning (matching concepts with the outdated model).
* Writes a file with concept annotations for later translation (tuning mode).
* Writes a file of new model concepts that have a skos:broader relation for further analysis (tuning mode).

**Key function:**  
parse\_xml\_to\_ttl\_from\_url(xml\_url, old\_model\_url, new\_model\_doc, new\_model\_url, ...) — the main data processing function.

## 2.2 Concept Preparation Module — logic\_prepair.py

* Extracts labels, descriptions, and annotations from XML concepts.
* Defines concept id, name, and description.
* Generates RDF triples for skos:Concept.
* Processes annotations (CONTEXT, RECOMMENDED\_REPRESENTATION, CODELIST\_ID).

**Key function:**  
prepair\_concepts(concept, namespaces, include\_context, codelist\_associations)

## 2.3 Concept Processing Module — logic\_triplets.py

This module processes concepts extracted from XML and determines their RDF relationships.

* Processes XML concepts and defines RDF relations.
* Determines types of relations (skos:broader, skos:narrower, skos:related).
* Aligns new model concepts with the outdated model (skos:exactMatch, skos:closeMatch).
* Uses templates (logic\_templates.py) for corrections and hierarchy adjustments.
* Generates RDF triple strings for new model concepts.
* If include\_context=True, adds additional CONTEXT annotation.
* Generates annotation strings for translation (tuning mode).

**Key function:**  
process\_concepts(concept, namespaces, include\_context, concepts, codelist\_associations, old\_graph)

## 2.4 RDF and String Utility Module — logic\_function.py

Provides helper functions for string and concept processing:

* Formats literals (format\_literal).
* Splits strings by separator (split\_by\_separator).
* Finds string differences (find\_first\_difference).
* Normalizes text (normalize\_text).
* Transforms concept identifiers (transform\_concept\_id).

## 2.5 Templates and Fixes — logic\_templates.py

* Dictionary of label fixes (LABEL\_FIXES).
* Defines hierarchy (BROADER\_FIXES).

Functions:

* get\_label\_fixes() — returns label corrections.
* get\_broader\_fixes() — returns predefined skos:broader relations.

## 2.6 Code List CSV Generator — cl\_concept\_writer.py

Writes associations of code lists and concepts into CSV.

**Key function:**  
write\_codelist\_csv(codelist\_output, codelist\_associations)

## 2.7 Tuning Module — tuning.py

Compares concept identifiers between outdated and new RDF models, detects mismatches and saves them.

**Key function:**  
compare\_ID(old\_model\_url, new\_ttl\_path, output\_file)

## 2.8 XML/RDF Loader and Parser — loadparser.py

Loads XML document and RDF graph of the outdated model.

**Key function:**  
loader\_parse\_xml\_ttl\_from\_url(xml\_url, old\_model\_url)

## 2.9 XML Data Extractor — xml\_extractor.py

Parses XML file and extracts concepts.

**Key function:**  
extract\_xml\_content(root, namespaces)

# Workflow

1. **Load XML and outdated model**  
   loader\_parse\_xml\_ttl\_from\_url loads XML document with concepts and RDF graph.
2. **Extract data from XML**  
   extract\_xml\_content extracts concept id, name, and description.
3. **Process concepts**  
   process\_concepts analyzes concepts and determines relations (skos:broader, skos:narrower, skos:related).
4. **Generate RDF model**  
   Creates a new skos:ConceptScheme, builds RDF triples
5. **Write to Turtle and CSV**The model is saved into a TTL file, and the code lists are written to CSV.
6. **Tuning and alignment**  
   During processing (when the tuning parameter is specified), files are generated for result analysis and for subsequent manual creation of correction dictionaries (LABEL\_FIXES) and exceptions (BROADER\_FIXES).

# Concept and Relation Processing Logic

## 4.1 Concept Definition

logic\_prepair.py analyzes XML concept structure and extracts:

* Identifier (id)
* Name
* Description
* Additional annotations (context, code lists, recommended formats)

Converted into RDF:

spr-concept:C1 a skos:Concept ;

rdfs:label "Concept One"@en ;

skos:definition "First concept description."@en ;

skos:notation "urn:example:C1" ;

skos:inScheme spr-concept:cog .

## 4.2 Defining Relations Between Concepts of the New Model

The module **logic\_triplets.py** analyzes the concepts of the new model and establishes their relations:

* **skos:broader** — broader concept
* **skos:narrower** — narrower concept (if the parameter *narrow\_include* is specified — reserved for future use)
* **skos:related** — related concept

The determination of hierarchical relations between concepts is based on the analysis of their labels, descriptions, and connections with other concepts. Within this process, a general logic is applied to identify relations: *broader* (broader concept), *narrower* (more specific concept), and *related* (associated concept). However, since automatic determination of these relations does not always yield correct results, tuning and adjustment mechanisms are used.

***General Logic for Establishing Relations***

1. **Semantic analysis of names and descriptions**
   * If the description of one concept contains the name of another concept, a *broader* or *narrower* relation may exist.
   * If a label contains a separator (e.g., " - "), the first part may indicate *broader* and the second part *narrower*.
2. **Analysis of annotations in XML**
   * If the XML document specifies related terms (**RELATED\_TERMS**) for a concept, they may be interpreted as *broader*, *narrower*, or *related* depending on the context.
3. **Use of the BROADER\_FIXES dictionary**
   * If a specific value is defined for a concept in the BROADER\_FIXES dictionary, the *broader* relation is set explicitly, and the general logic is not applied.
   * If the value is empty, the concept retains only *related* relations.

## 4.3 Tuning Based on Concept Label Corrections

During concept processing, tuning files are analyzed and discrepancies are identified. Based on these, a dictionary **LABEL\_FIXES** is created, containing label corrections.  
This is necessary due to possible typos, changes in terminology, and inconsistencies between models.

**Examples of corrections:**

* *timelinesst* → *timeliness* (typo correction)
* *coherence - cross-domain* → *coherence - cross domain* (format standardization)
* *relevance - user satisifaction* → *relevance - user satisfaction* (spelling correction)

**Example of the LABEL\_FIXES dictionary (logic\_templates.py):**

LABEL\_FIXES = {

"timelinesst": "timeliness",

"coherence - cross-domain": "coherence - cross domain",

"relevance - user satisifaction": "relevance - user satisfaction"

}

If a concept contains the typo *"timelinesst"*, it is automatically replaced with *"timeliness"*.

## 4.4 Tuning of Concept Hierarchy

During concept processing, tuning files are analyzed and discrepancies are identified. Based on these, a dictionary **BROADER\_FIXES** (logic\_templates.py) is created, defining strict skos:broader relations for concepts.

**Example:**

BROADER\_FIXES = {

"DSD": "DATA\_SET",

"ATTRIBUTE": "DSD",

"DIMENSION": "DSD",

"MEASURE": "DSD",

"ORGANISATION\_UNIT": "CONTACT"

}

Thus, if the concept "ATTRIBUTE" does not have an explicit skos:broader relation, it is forcibly assigned to "DSD", while all other relations remain skos:related.

## 4.5 Alignment of Concepts Between the New and Outdated Models

The alignment of concepts from the new model with those of the old model is carried out to determine semantic correspondences. Both automatic text analysis and additional identifier comparison are applied.

**General Matching Logic:**

1. **Exact match of normalized names and labels or descriptions:**
   * If the normalized names (identifiers) and the label (or description) of concepts in the new and old models coincide, a skos:exactMatch relation is established.
2. **Partial match:**
   * If labels or the description of a concept in the new model fully or partially coincide with those of a concept in the old model, a skos:closeMatch relation is established.

The **tuning.py** module performs additional alignment of concepts between the new and outdated models according to the following criteria (without writing new model triples):

* skos:exactMatch — set if the concept identifier matches the header comment (#) in the outdated model, and if the names also coincide.
* **skos:closeMatch** — set if the concept names are similar but the identifiers differ.

If the concept identifiers in the new and outdated models (comment before the concept, like # ID) coincide but the names do not, the triples of the concept are written into a separate file no\_match\_concepts.ttl.

# Conclusion

The project provides a comprehensive solution for updating the SDMX glossary, including:

* Extraction of concepts from XML.
* Formation of an RDF model with skos:broader, skos:related, skos:exactMatch.
* Error correction and hierarchy adjustments of concepts.
* Alignment with the outdated model and detection of inconsistencies.