VocBench Sheet2RDF import guide

Version : 1.0
Date : 2025-10-01
Abstract : *This document gives practical procedures and use-cases to import tabular CSV or spreadsheet data into VocBench using the Sheet2RDF tool.*
Authors : thomas.francart@sparna.fr, marie.muller@sparna.fr

Introduction

This document gives step-by-step procedures on how to use VocBench Sheet2RDF tool to import data in Vocbench projects.

This document aims at being practical and use-case oriented, leaving aside the theoretical details.

For more information, please refer to the official Sheet2RDF website as well as to the dedicated page of the Sheet2RDF integration in VocBench. You can also check out the Sheet2RDF tutorial page.

Prerequisites

- You need to be logged in a VocBench project with an account who is either a project manager or an administrator of the project, since they are the only profiles allowed to work with Sheet2RDF.
- A spreadsheet with your data organized in columns, one property field per column. (a good practice is to edit the column headers with the correct property names before loading the spreadsheet, see further details below in the chapter Automated : mapping based on header names)
- In case we want to import SKOS Concepts, as for example a simple controlled list (doc types, statuses, etc.), you will need to create a project with a ConceptScheme with the ConceptScheme ID of the vocabulary and its human readable name.

Data sheet structure prerequisites

1. The spreadsheet tabs content has to be clean, and the first line must contain the title of the column. No other lines should be present before the header.
2. Also make sure there are no blank lines in the middle of the file.
3. Each cell in the file must contain only one value. If your file contains multiple values, you need to separate them into several columns with the same header : for example it is perfectly valid to have multiple columns with the same header “skos:altLabel”, each containing a single synonym value.
4. The spreadsheet can contain multiple tabs
5. Save your file in xlsx, odt, or csv format.

Prepare the vocabulary project in VocBench

The data you will import with Sheet2RDF has to be stored in a vocabulary project in VocBench.

If you don't have the project yet, start by creating it. To do so, you can follow the corresponding VocBench documentation.

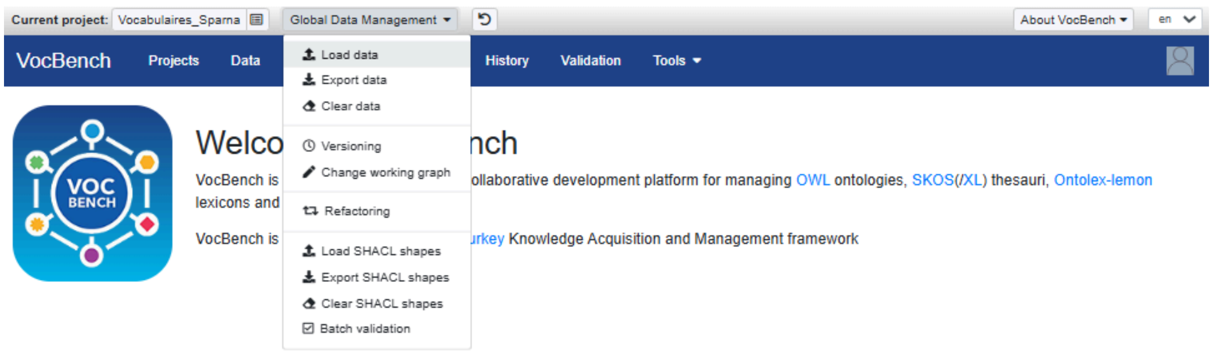
The project can be either a SKOS project or an OWL project. If the import you want to make is an import of SKOS Concepts to populate a new ConceptScheme, make sure to manually create the ConceptScheme before, by checking the corresponding section in VocBench documentation.

Alternatives to Sheet2RDF

Direct RDF loading

As a reminder, Sheet2RDF is not the only way to load data in VocBench. It is directly possible to load existing SKOS or RDF files inside a project.

To load already existing SKOS or RDF files, go to the upper drop-down menu and choose the « Load data » feature.



Welco[...]nch

VocBench is [...]ollaborative development platform for managing OWL ontologies, SKOS(/XL) thesauri, Ontolex-lemon lexicons and [...]

VocBench is [...]urkey Knowledge Acquisition and Management framework

A dedicated window opens where you can configure the source of your data, select a file, and specify the base URI and the import procedure. For more information, refer to the official VocBench "load data" documentation.

xls2rdf

From an Excel file, you can also use Sparna's Excel-to-RDF converter tool : to generate an RDF file from an Excel spreadsheet structured in a specific way.

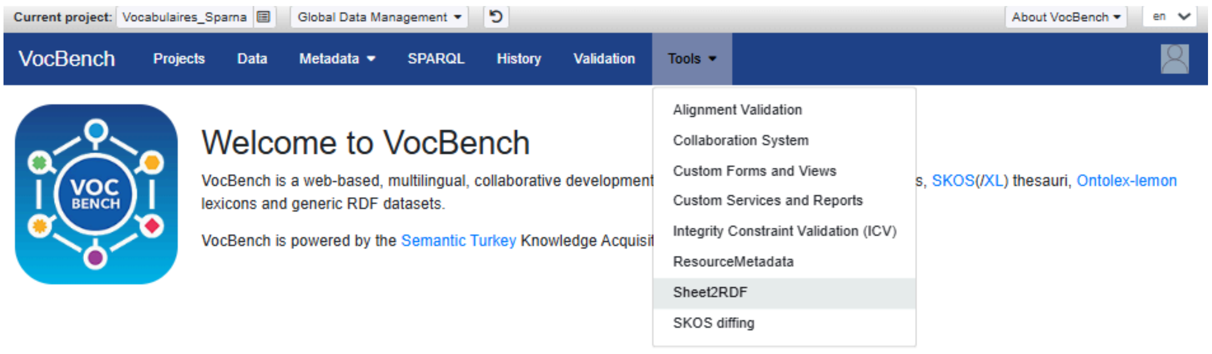
The tool can produce any RDF, not only SKOS (lists of foaf:Person, of schema:Event, SHACL, OWL, etc.).

The main difference between xls2rdf and Sheet2RDF is that xls2rdf relies entirely on the headers configuration, no external mapping file is necessary.

Import data using VocBench Sheet2RDF tool

The Sheet2RDF screen and workflow

To access Sheet2RDF, go to the menu “Tools”, and click on “Sheet2RDF”.

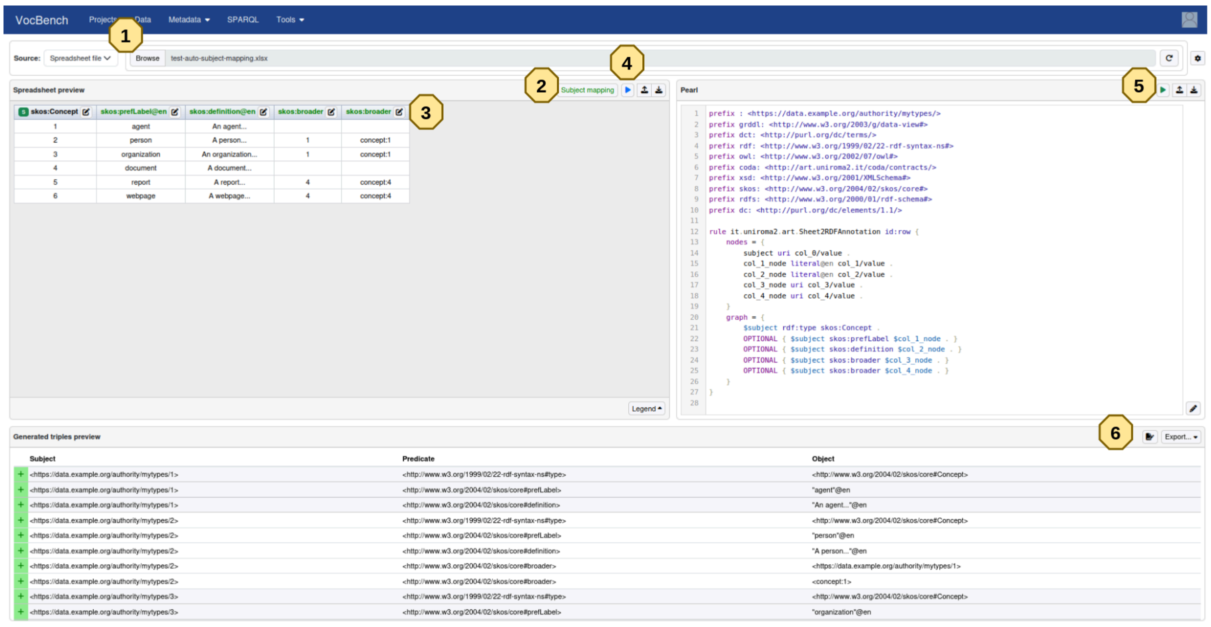


Welcome to VocBench

VocBench is a web-based, multilingual, collaborative development
s, SKOS(/XL) thesauri, Ontolex-lemon
lexicons and generic RDF datasets.

VocBench is powered by the Semantic Turkey Knowledge Acquisit

The import workflow involves the following steps in the screen:



Source: Spreadsheet file
Browse test-auto-subject-mapping.xlsx

Spreadsheet preview

skos:Concept	skos:prefLabel@en	skos:definition@en	skos:broader	skos:broader
1	agent	An agent...		
2	person	A person...	1	concept:1
3	organization	An organization...	1	concept:1
4	document	A document...		
5	report	A report...	4	concept:4
6	webpage	A webpage...	4	concept:4

Pearl

```
prefix : <https://data.example.org/authority/mytypes/>
prefix grddl: <http://www.w3.org/2003/g/data-view#>
prefix dct: <http://purl.org/dc/terms/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix coda: <http://art.uniroma2.it/coda/contracts/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix skos: <http://www.w3.org/2004/02/skos/core#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix dc: <http://purl.org/dc/elements/1.1/>

rule it.uniroma2.art.Sheet2RDFAnnotation id:row {
    nodes = {
        subject uri col_0/value .
        col_1_node literal@en col_1/value .
        col_2_node literal@en col_2/value .
        col_3_node uri col_3/value .
        col_4_node uri col_4/value .
    }
    graph = {
        $subject rdf:type skos:Concept .
        OPTIONAL { $subject skos:prefLabel $col_1_node . }
        OPTIONAL { $subject skos:definition $col_2_node . }
        OPTIONAL { $subject skos:broader $col_3_node . }
        OPTIONAL { $subject skos:broader $col_4_node . }
    }
}

```

Generated triples preview

	Subject	Predicate	Object
+	<https://data.example.org/authority/mytypes/1>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core#Concept>
+	<https://data.example.org/authority/mytypes/1>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"agent"@en
+	<https://data.example.org/authority/mytypes/1>	<http://www.w3.org/2004/02/skos/core#definition>	"An agent..."@en
+	<https://data.example.org/authority/mytypes/2>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core#Concept>
+	<https://data.example.org/authority/mytypes/2>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"person"@en
+	<https://data.example.org/authority/mytypes/2>	<http://www.w3.org/2004/02/skos/core#definition>	"A person..."@en
+	<https://data.example.org/authority/mytypes/2>	<http://www.w3.org/2004/02/skos/core#broader>	<https://data.example.org/authority/mytypes/1>
+	<https://data.example.org/authority/mytypes/2>	<http://www.w3.org/2004/02/skos/core#broader>	<concept:1>
+	<https://data.example.org/authority/mytypes/3>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core#Concept>
+	<https://data.example.org/authority/mytypes/3>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"organization"@en

1. **Load** the spreadsheet
2. Define the **Subject mapping** with the “Subject mapping” button
3. Define **each column mapping** with the edit icon next to each column header
4. Generate the **PEARL technical mapping** definition
    a. Here, the PEARL mapping definition can be manually edited if needed
5. **Trigger the actual conversion** of the spreadsheet file from the PEARL definition
6. Either **download** the resulting RDF data, or **store** it in the current VocBench project

Prepare your data sheet

If you can modify your spreadsheet file, edit the column headers in the first line to match the data model as configured in VocBench for your project. This will make the mapping step much easier as Sheet2RDF will be able to guess the mapping from the column headers.

Follow the rules below to edit your headers, and refer to the Sheet2RDF complete heuristics documentation for more information.

Use property URI identifiers as column names (e.g. “skos:prefLabel”)

Names for headers can be a valid property name (e.g. "`skos:prefLabel`")

For this name to be meaningfully used, its prefix should be mappable to a given namespace, by means of one of the following:

1. the prefix is known a priori by VocBench, in the project where the import is done (i.e. skosxl, skos, owl, rdfs, rdf,...)
2. prefix/namespace is defined in the `prefix_mapping` sheet. You can add a sheet called “prefix_mapping” to your spreadsheet, with 2 columns : the prefix name in the first column (e.g. “rico”), and the prefix URI in the second column (e.g. “https://www.ica.org/standards/RiC/ontology#”). This should start at the first line of the sheet.

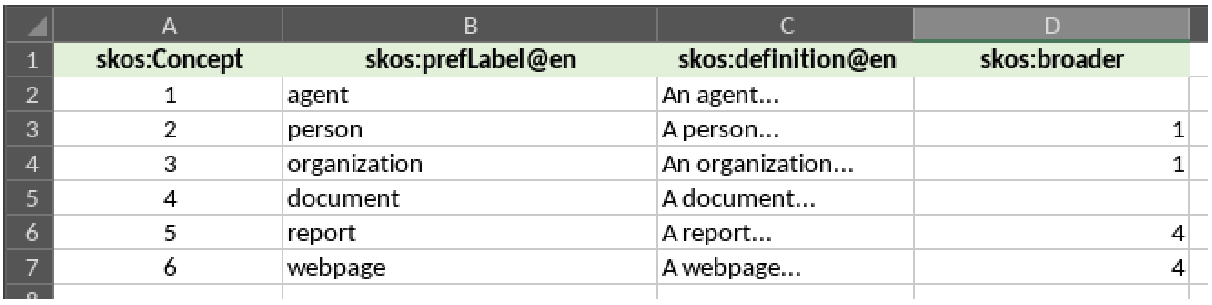
Optionally indicate a language tag with @xx

A language tag can be added at the end of the header with “@xx.”, for example “`skos:prefLabel@en`”.

Optionally use a class URI identifier as the name for a single column (e.g. “skos:Concept”)

A class URI can also be used to mark the column corresponding to the subject (e.g. “`foaf:Person`”). This should be done on a single column only, in the case a column contains some identifiers that can be used to create the subject URI of the triples (see below the “Subject mapping” section).

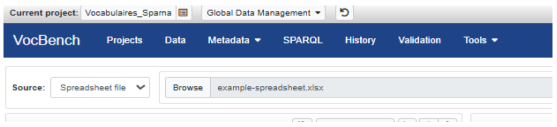
An example of how an input Excel file can look like to be automatically mapped is the following:



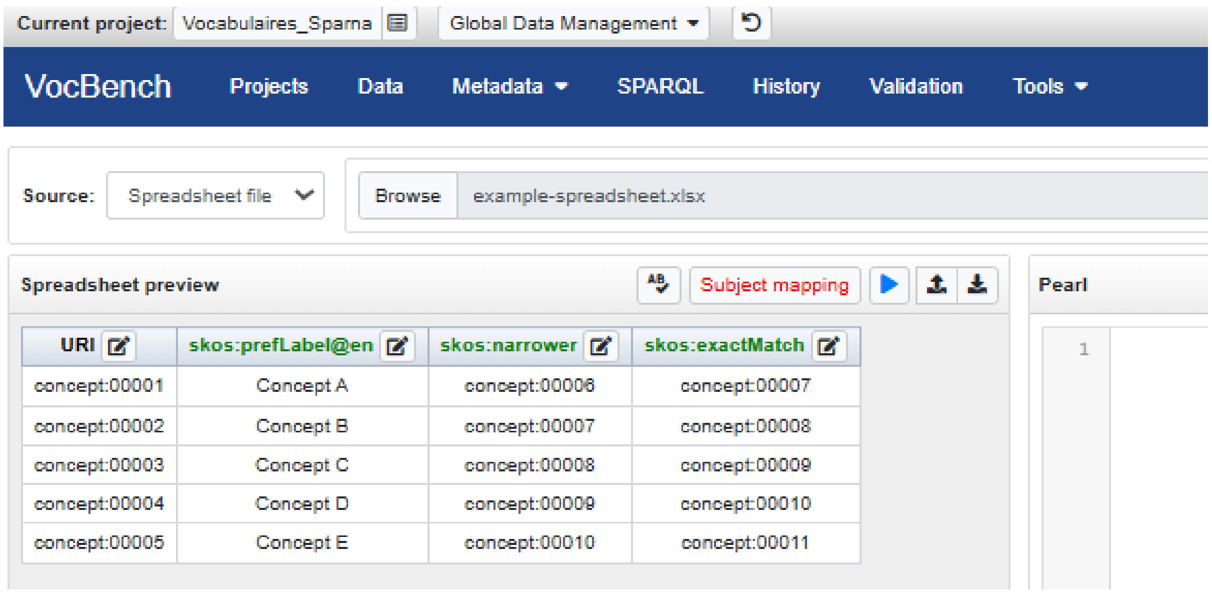
	A	B	C	D
1	skos:Concept	skos:prefLabel@en	skos:definition@en	skos:broader
2	1	agent	An agent...	
3	2	person	A person...	1
4	3	organization	An organization...	1
5	4	document	A document...	
6	5	report	A report...	4
7	6	webpage	A webpage...	4

Load your file in Sheet2RDF

Once in the Sheet2RDF screen, load your Spreadsheet file by clicking on Sheet2RDF browse button :



Your spreadsheet appears on screen and the mapping can be completed via Sheet2RDF's mapping wizard :

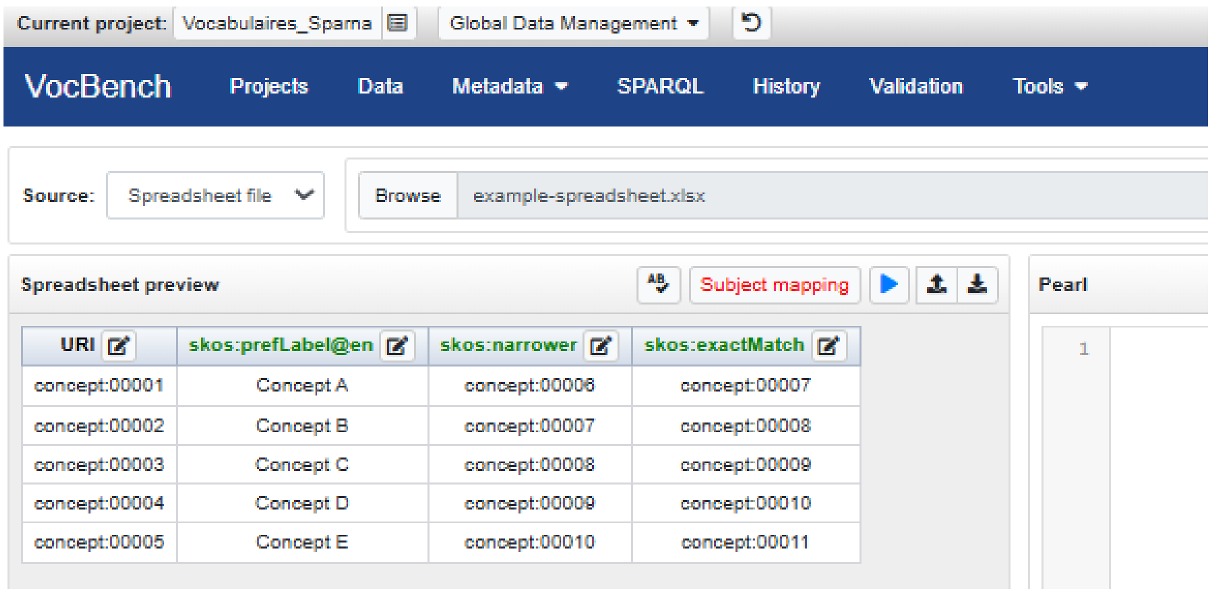


URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
concept:00001	Concept A	concept:00006	concept:00007
concept:00002	Concept B	concept:00007	concept:00008
concept:00003	Concept C	concept:00008	concept:00009
concept:00004	Concept D	concept:00009	concept:00010
concept:00005	Concept E	concept:00010	concept:00011

The Mapping wizard

Automated : mapping based on header names

Have a look at the headers of your columns :



URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
concept:00001	Concept A	concept:00006	concept:00007
concept:00002	Concept B	concept:00007	concept:00008
concept:00003	Concept C	concept:00008	concept:00009
concept:00004	Concept D	concept:00009	concept:00010
concept:00005	Concept E	concept:00010	concept:00011

If you named the column headers according to the rules described earlier, the system will automatically recognize its properties and instantly validate the mapping when loading your spreadsheet. The columns that have been matched automatically will appear with a **green** header.

If you also used a class URI to indicate the subject column, the subject mapping will also appear in green. Otherwise you need to proceed to the subject mapping (see below) in order to assign a subject (a URI, from your URI column) to each line of your spreadsheet.

Wizard : using wizard screens to create mappings for each column

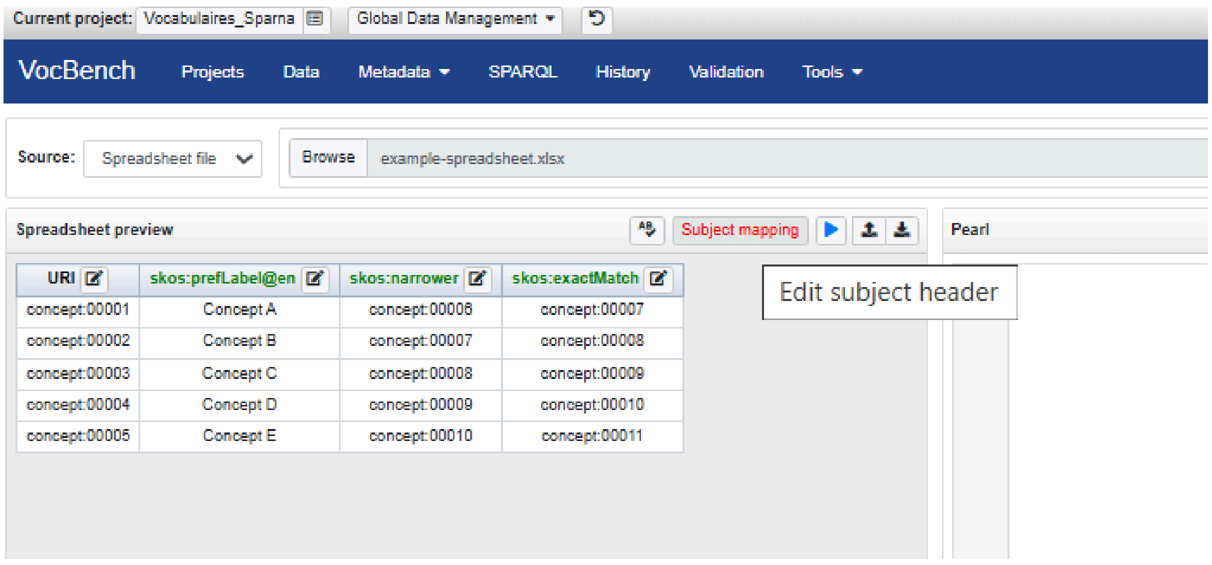
Let's start the manual mapping of your file via the mapping wizard. The mapping will be achieved in two main steps :

1. The **subject mapping**, to identify the items of the spreadsheet as the subjects of the triples to be added to the graph, using the available URIs or creating some on purpose
2. The **columns mapping**, to convert each column into a triple with a predicate (ex : skos:prefLabel, skos:exactMatch, etc.), combining a « node » that represents the content of the spreadsheet cell with its projection in the graph (or « graph application »).

Start the subject mapping using the Subject Header editor

Any triple that will be added to the graph must have a subject, this is the URI that identifies every item listed in the spreadsheet. In this concern, Sheet2RDF needs to know how to generate or assign the subject (the URI) for each line. This is described in the subject mapping section of the Sheet2RDF documentation.

This can be done using the Subject Header editor (the « subject mapping » button that appears in red) just above the spreadsheet preview :

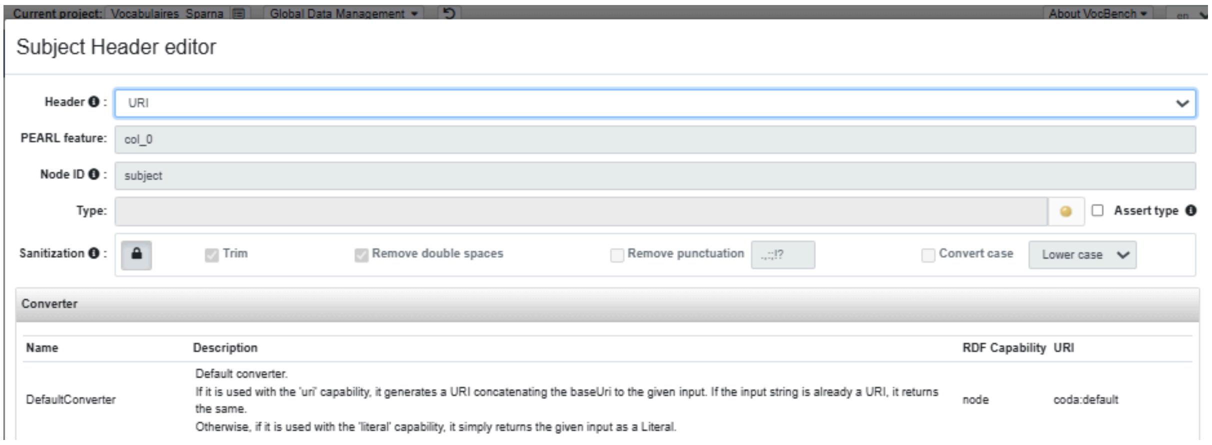


Source: Spreadsheet file — example-spreadsheet.xlsx

Spreadsheet preview

URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
concept:00001	Concept A	concept:00006	concept:00007
concept:00002	Concept B	concept:00007	concept:00008
concept:00003	Concept C	concept:00008	concept:00009
concept:00004	Concept D	concept:00009	concept:00010
concept:00005	Concept E	concept:00010	concept:00011

Click on the « subject mapping » button to start the subject mapping. The Subject Header Editor appears where you can configure the subject mapping :



Subject Header editor

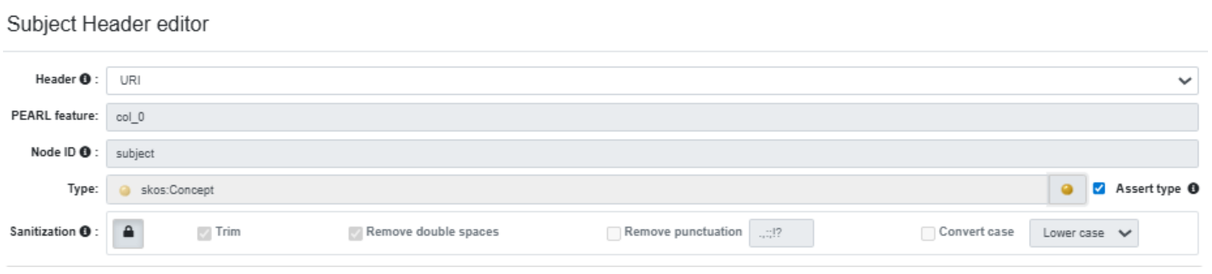
Header: URI
PEARL feature: col_0
Node ID: subject
Type:
Assert type
Sanitization: Trim ✓ · Remove double spaces ✓ · Remove punctuation .,;:!? · Convert case · Lower case

Converter

Name	Description	RDF Capability	URI
DefaultConverter	Default converter. If it is used with the 'uri' capability, it generates a URI concatenating the baseUri to the given input. If the input string is already a URI, it returns the same. Otherwise, if it is used with the 'literal' capability, it simply returns the given input as a Literal.	node	coda:default

Select the subject column and the type

- Select the column from your file from which the subject mapping will be determined. Depending on the cases, this can be a column containing a string identifier, a complete URI identifier, or a label (see below)
- You can also assign a type (`rdf:type`) to every subject that will be created in the import. For this, check the “Assert type” box and select the class that will be assigned as a type to every subject. Typically, in SKOS projects, you would select “skos:Concept” to populate a controlled vocabulary :



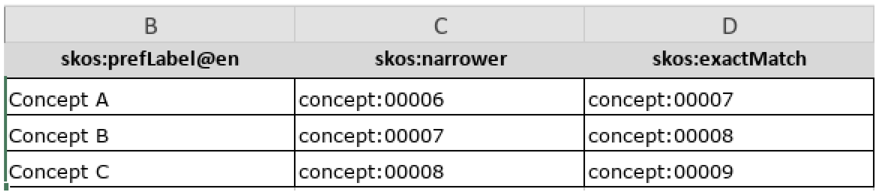
Subject Header editor

Header:	URI
PEARL feature:	col_0
Node ID:	subject
Type:	skos:Concept ✓ Assert type
Sanitization:	✓ Trim ✓ Remove double spaces ☐ Remove punctuation ,.;:!? ☐ Convert case Lower case

Go to the Converter section below to select the converter you need. The converter is responsible for generating the URI of the subject from the column content.

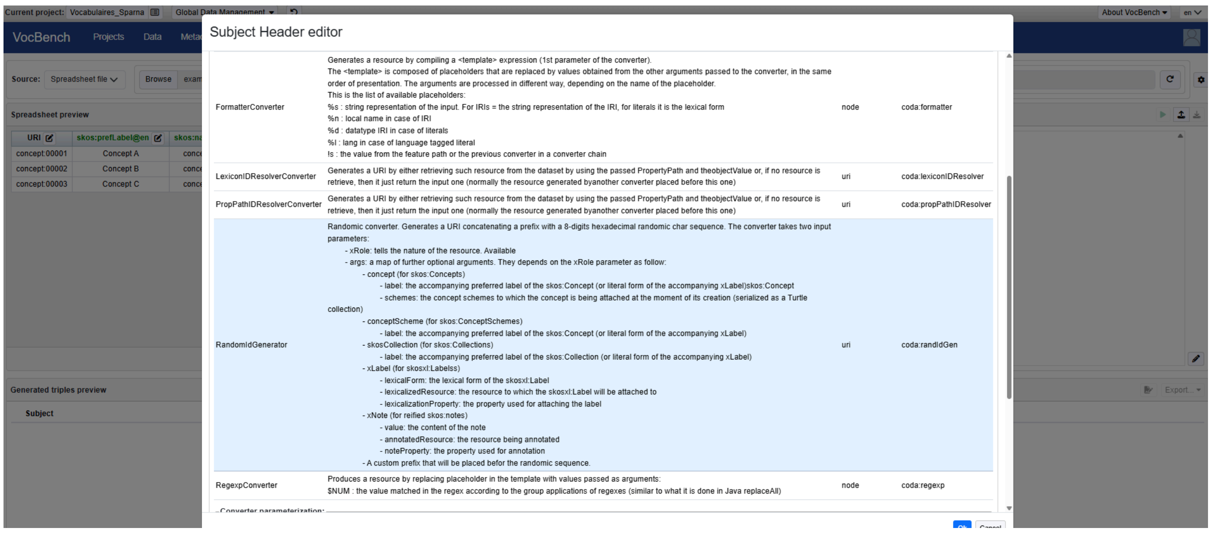
Case 1 : the table does not contain any usable identifier

In case the table does not contain any usable identifier, select “**RandomIdGenerator**”. For example:



B	C	D
skos:prefLabel@en	skos:narrower	skos:exactMatch
Concept A	concept:00006	concept:00007
Concept B	concept:00007	concept:00008
Concept C	concept:00008	concept:00009

“`RandomIdGenerator`” generates a random URI and assigns it to the subject node.

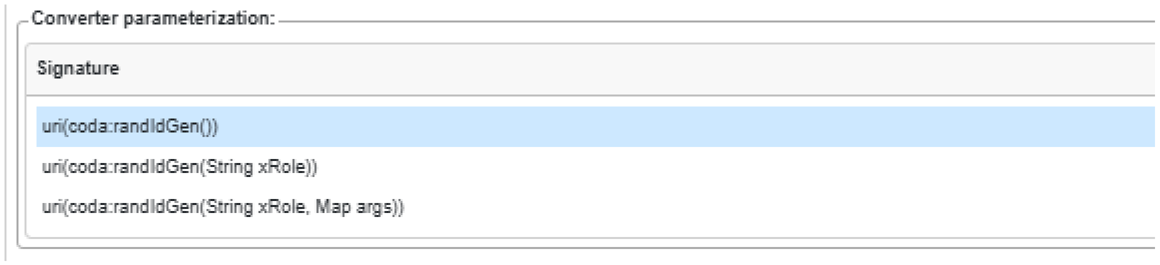


Subject Header editor

FormatterConverter	Generates a resource by compiling a <template> expression (1st parameter of the converter). The <template> is composed of placeholders that are replaced by values obtained from the other arguments passed to the converter, in the same order of presentation. The arguments are processed in different way, depending on the name of the placeholder. This is the list of available placeholders: %s : string representation of the input. For IRIs = the string representation of the IRI, for literals it is the lexical form %n : local name in case of IRI %d : datatype IRI in case of literals %l : lang in case of language tagged literal !s : the value from the feature path or the previous converter in a converter chain	node	coda:formatter
LexiconIDResolverConverter	Generates a URI by either retrieving such resource from the dataset by using the passed PropertyPath and theobjectValue or, if no resource is retrieve, then it just return the input one (normally the resource generated byanother converter placed before this one)	uri	coda:lexiconIDResolver
PropPathIDResolverConverter	Generates a URI by either retrieving such resource from the dataset by using the passed PropertyPath and theobjectValue or, if no resource is retrieve, then it just return the input one (normally the resource generated byanother converter placed before this one)	uri	coda:propPathIDResolver
RandomIdGenerator	Randomic converter. Generates a URI concatenating a prefix with a 8-digits hexadecimal randomic char sequence. The converter takes two input parameters: - xRole: tells the nature of the resource. Available - args: a map of further optional arguments. They depends on the xRole parameter as follow: - concept (for skos:Concepts) - label: the accompanying preferred label of the skos:Concept (or literal form of the accompanying xLabel)skos:Concept - schemes: the concept schemes to which the concept is being attached at the moment of its creation (serialized as a Turtle collection) - conceptScheme (for skos:ConceptSchemes) - label: the accompanying preferred label of the skos:Concept (or literal form of the accompanying xLabel) - skosCollection (for skos:Collections) - label: the accompanying preferred label of the skos:Collection (or literal form of the accompanying xLabel) - xLabel (for skosxl:Labelss) - lexicalForm: the lexical form of the skosxl:Label - lexicalizedResource: the resource to which the skosxl:Label will be attached to - lexicalizationProperty: the property used for attaching the label - xNote (for reified skos:notes) - value: the content of the note - annotatedResource: the resource being annotated - noteProperty: the property used for annotation - A custom prefix that will be placed befor the randomic sequence.	uri	coda:randIdGen
RegexpConverter	Produces a resource by replacing placeholder in the template with values passed as arguments: $NUM : the value matched in the regex according to the group applications of regexes (similar to what it is done in Java replaceAll)	node	coda:regexp

Converter parameterization:

Below the converter selection you can select the “Signature” of the converter, with 3 possible choices:


Converter parameterization:

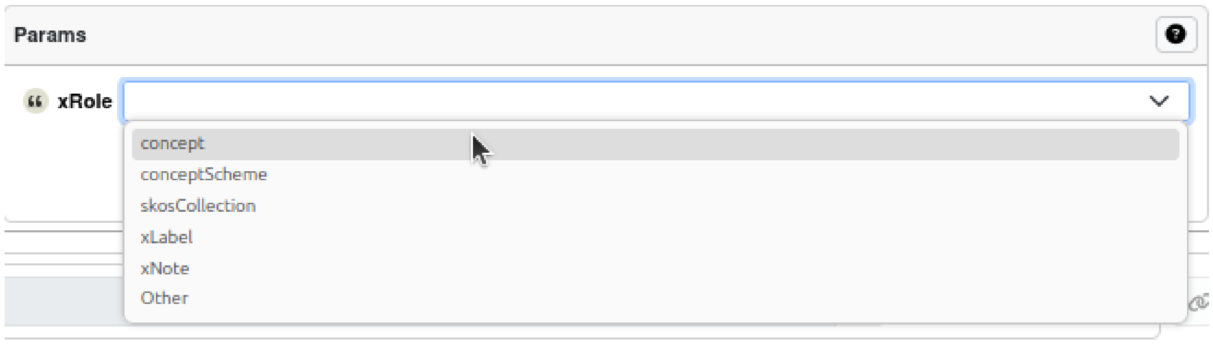
Signature

uri(coda:randIdGen())

uri(coda:randIdGen(String xRole))

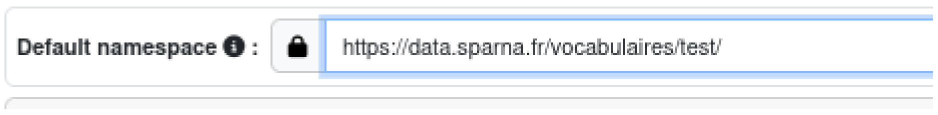
uri(coda:randIdGen(String xRole, Map args))

1. With the first choice "`uri(coda:randIdGen())`", the generator will produce URIs with a serial of eight random numbers, preceded by the "`undetermined`" prefix, such as <https://data.sparna.fr/vocabulaires/test#undetermined_9ab40489>
2. With the second choice "`uri(coda:randIdGen(String xRole))`", the generator will produce URI with a specified "role". We recommend to use this signature. The role needs to be selected as an argument:



   The "roles" are the ones known by VocBench, and cannot be changed. Typically you would choose "concept" when importing Concepts in a SKOS project, or "individual" when importing individuals in an OWL project. The generated URI will look like <https://data.sparna.fr/vocabulaires/test#c_9ab40489> ("c" corresponds to the prefix for the "concept" role).
3. More advanced options can be provided with the third choice "`uri(coda:randIdGen(String xRole, Map args))`", but this is for advanced usage

The generated URI will always use, by default, the base URI of the project as the beginning of the URI. You can adjust this by modifying the "Default namespace" parameter of the generator:

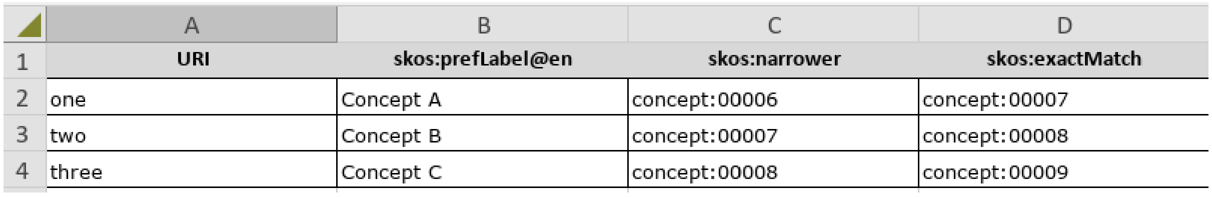

Default namespace : https://data.sparna.fr/vocabulaires/test/

Don't forget the final "#" or "/" in this case.

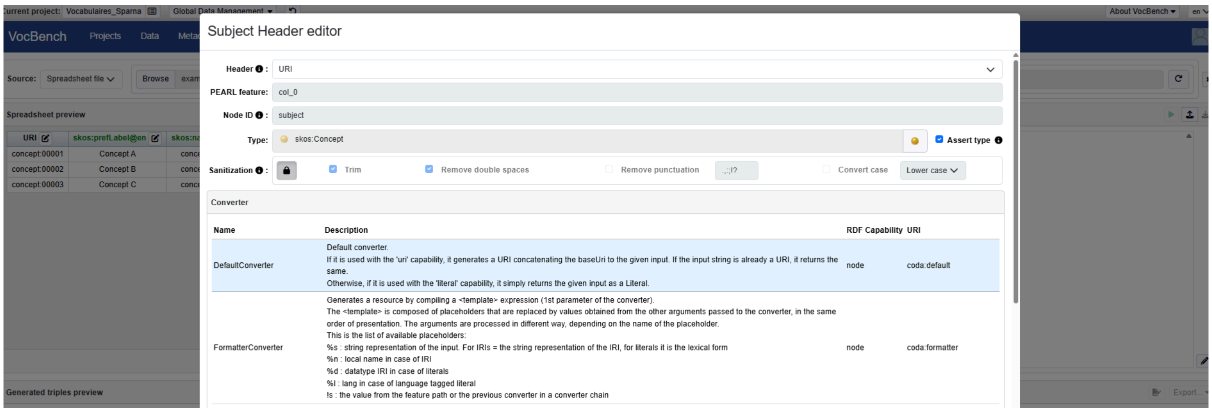
Case 2 : the table contains an existing unique identifier (numerical or alphabetical)

In case when the table contains an existing unique identifier (numerical or alphabetical), select "**`DefaultConverter`**".

For example:



	A	B	C	D
1	URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
2	one	Concept A	concept:00006	concept:00007
3	two	Concept B	concept:00007	concept:00008
4	three	Concept C	concept:00008	concept:00009



Subject Header editor

- Header: URI
- PEARL feature: col_0
- Node ID: subject
- Type: skos:Concept — [✓] Assert type
- Sanitization: [✓] Trim · [✓] Remove double spaces · [ ] Remove punctuation .,:;!? · [ ] Convert case Lower case

Converter

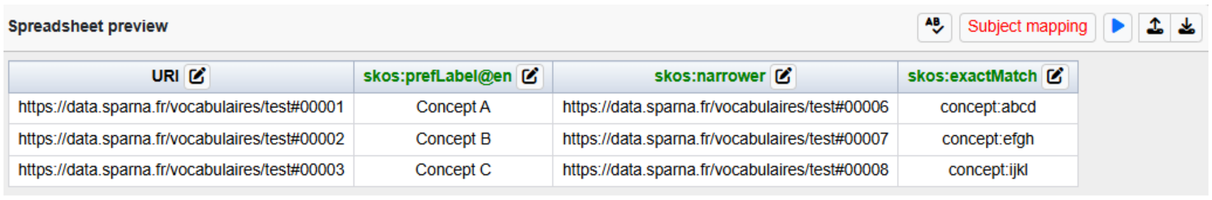
Name	Description	RDF Capability	URI
DefaultConverter	Default converter. If it is used with the 'uri' capability, it generates a URI concatenating the baseUri to the given input. If the input string is already a URI, it returns the same. Otherwise, if it is used with the 'literal' capability, it simply returns the given input as a Literal.	node	coda:default
FormatterConverter	Generates a resource by compiling a <template> expression (1st parameter of the converter). The <template> is composed of placeholders that are replaced by values obtained from the other arguments passed to the converter, in the same order of presentation. The arguments are processed in different way, depending on the name of the placeholder. This is the list of available placeholders: %s : string representation of the input. For IRIs = the string representation of the IRI, for literals it is the lexical form %n : local name in case of IRI %d : datatype IRI in case of literals %l : lang in case of language tagged literal !s : the value from the feature path or the previous converter in a converter chain	node	coda:formatter

"`DefaultConverter`" will concatenate the content of the column to the base URI of the project (you can modify this base URI by adjusting the "Default namespace" option at the bottom).

Case 3 : the table already contains a complete URI identifier

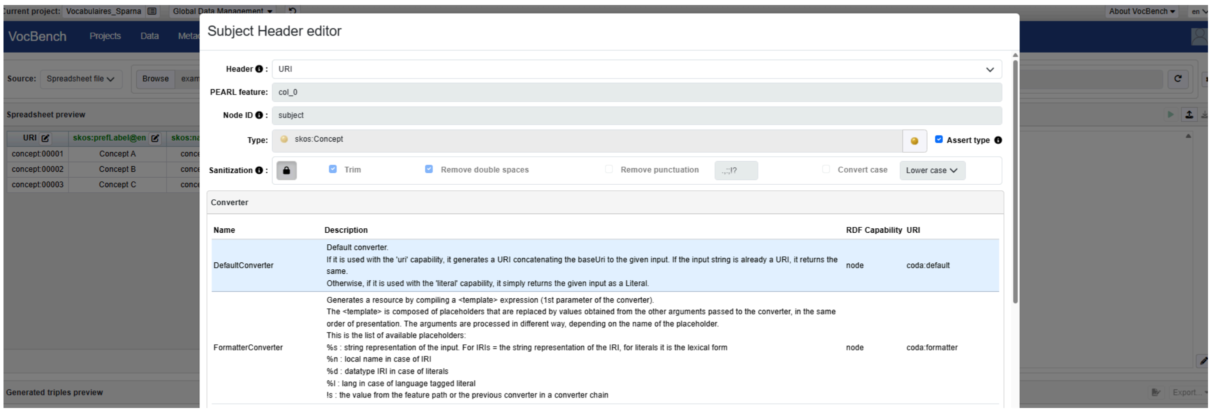
In case when the table already contains a complete URI identifier, select "**`DefaultConverter`**".

For example:



Spreadsheet preview

URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
https://data.sparna.fr/vocabulaires/test#00001	Concept A	https://data.sparna.fr/vocabulaires/test#00006	concept:abcd
https://data.sparna.fr/vocabulaires/test#00002	Concept B	https://data.sparna.fr/vocabulaires/test#00007	concept:efgh
https://data.sparna.fr/vocabulaires/test#00003	Concept C	https://data.sparna.fr/vocabulaires/test#00008	concept:ijkl


Subject Header editor

Header:	URI
PEARL feature:	col_0
Node ID:	subject
Type:	skos:Concept ☑ Assert type
Sanitization:	☑ Trim ☑ Remove double spaces ☐ Remove punctuation .,:;!? ☐ Convert case Lower case

Converter

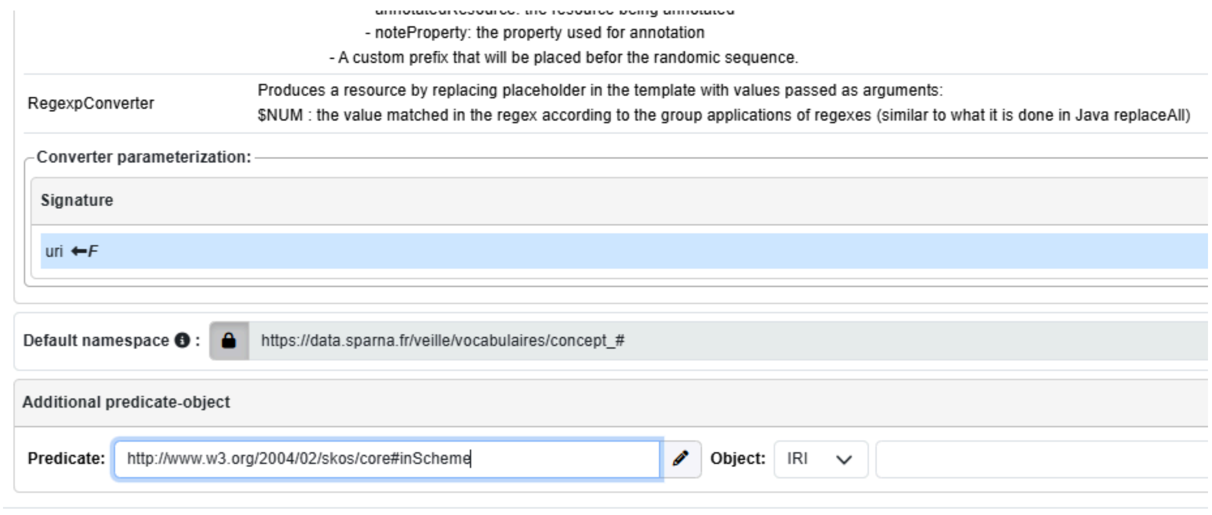
Name	Description	RDF Capability	URI
DefaultConverter	Default converter. If it is used with the 'uri' capability, it generates a URI concatenating the baseUri to the given input. If the input string is already a URI, it returns the same. Otherwise, if it is used with the 'literal' capability, it simply returns the given input as a Literal.	node	coda:default
FormatterConverter	Generates a resource by compiling a <template> expression (1st parameter of the converter). The <template> is composed of placeholders that are replaced by values obtained from the other arguments passed to the converter, in the same order of presentation. The arguments are processed in different way, depending on the name of the placeholder. This is the list of available placeholders: %s : string representation of the input. For IRIs = the string representation of the IRI, for literals it is the lexical form %n : local name in case of IRI %d : datatype IRI in case of literals %l : lang in case of language tagged literal !s : the value from the feature path or the previous converter in a converter chain	node	coda:formatter

In this case the converter will simply use the provided URI in the column as the subject URI.

The additional predicate-object

Finally, a section at the bottom allows the definition of additional predicate-object pairs to relate to the subject resource.

As an example, for a SKOS project, if we want to add the Additional predicate-object » « `skos:inScheme` » to link the items of the file to the ConceptScheme of the vocabulary, we need to specify the relation there. In this case, select “`skos:inScheme`” property in the Predicate field :


- noteProperty: the property used for annotation
- A custom prefix that will be placed befor the randomic sequence.

RegexpConverter	Produces a resource by replacing placeholder in the template with values passed as arguments: $NUM : the value matched in the regex according to the group applications of regexes (similar to what it is done in Java replaceAll)

Converter parameterization:

Signature

uri ←F

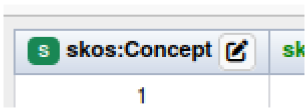
Default namespace: https://data.sparna.fr/veille/vocabulaires/concept_#

Additional predicate-object

Predicate: http://www.w3.org/2004/02/skos/core#inScheme Object: IRI

Then pick up the ConceptScheme URI of your vocabulary to relate to the subject resources.

Once the Subject mapping is done, the Subject mapping button turns green and a green dot with an [s] appears in the header of the column containing the identifiers. This means that this column contains the subjects of the triples that will be generated:



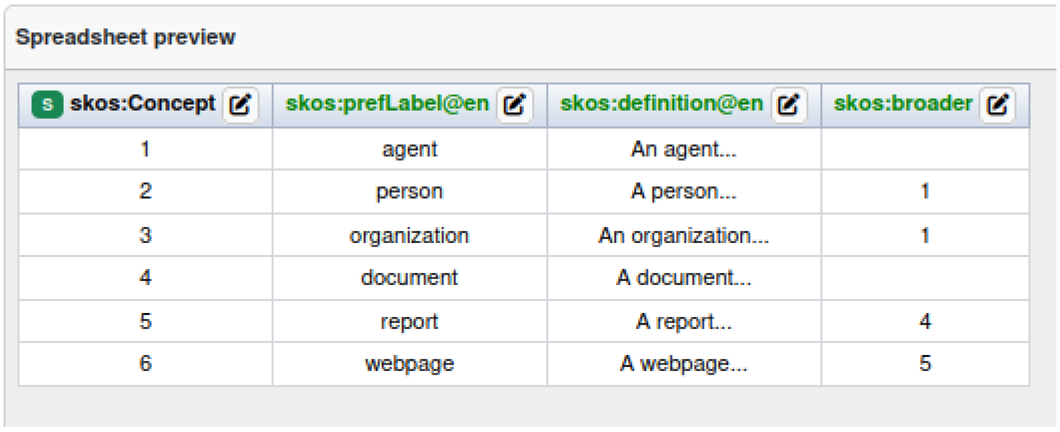
Define each column mapping

Now, you need to specify how each column will be mapped.

If the column naming rules have been followed !

If the column naming rules described earlier have been followed, the mapping will be automatic and you will see the columns in green.

For example, consider the following table:



Spreadsheet preview

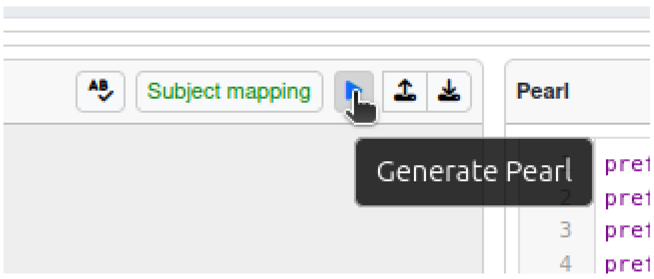
S skos:Concept	skos:prefLabel@en	skos:definition@en	skos:broader
1	agent	An agent...	
2	person	A person...	1
3	organization	An organization...	1
4	document	A document...	
5	report	A report...	4
6	webpage	A webpage...	5

This table has been mapped fully automatically by Sheet2RDF.

Note how:

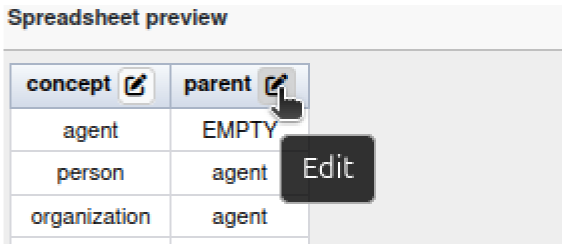
1. It uses `skos:Concept` as the header indicating the subject column on a column giving the identifier of each entry
2. It uses property URI `skos:prefLabel`, `skos:definition`, `skos:broader` for the other columns headers
3. It uses “`@en`” to indicate the language for the preferred label and the definition

In such a case, you can directly generate the PEARL mapping by clicking the blue arrow on the top right corner of the subject mapping panel:



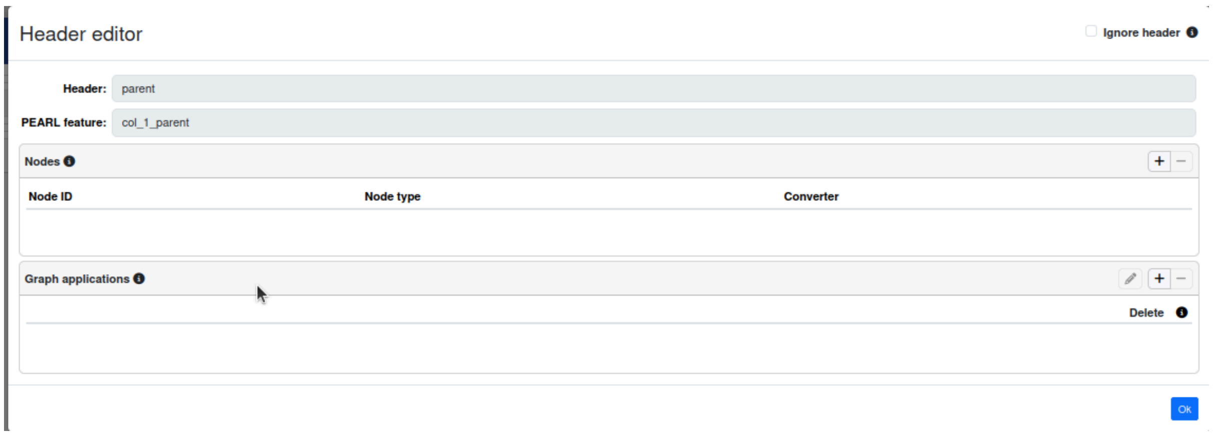
Otherwise follow the steps below to configure the mapping columns by columns.

To map a column, click on the pencil icon in the column header:



concept	parent
agent	EMPTY
person	agent
organization	agent

The Header Mapping Editor window will open:



Header: parent
PEARL feature: col_1_parent

- “Header” contains the name of the column you are working on
- “PEARL feature” contains the name of the associated variable that will be used in the Pearl script.

These are automatically assigned, there is no need to modify these 2 fields.

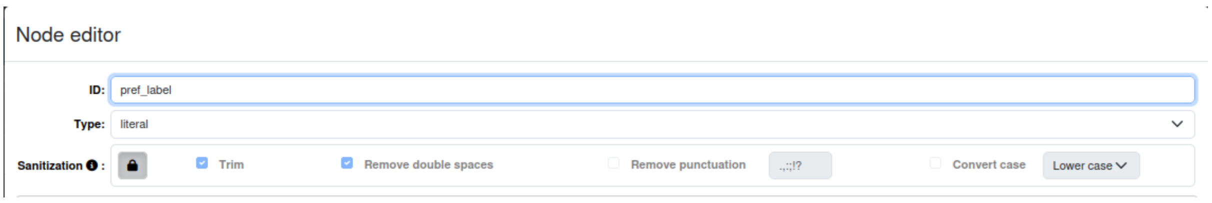
Logic: node and graph application

The header mapping configuration is done in two steps:

1. in the “Nodes” section, you will specify a procedure to convert the column data into an RDF value
2. in the “Graph Applications” section, you will indicate how the final triple(s) will be constructed, typically by simply selecting a predicate to refer to the node constructed in the first step.

Create the node

Click on the “+” in the “Nodes” panel to create a new node. This will display the node editor form:


Node editor

ID: pref_label

Type: literal

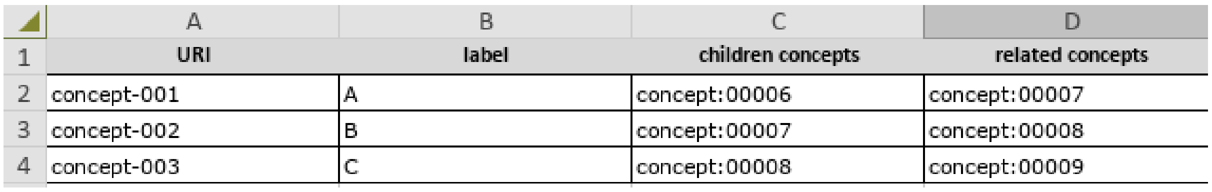
Sanitization: [✓ Trim] [✓ Remove double spaces] Remove punctuation .,;:!? Convert case Lower case

1. Enter an ID for the node, this can be any variable name, e.g. “pref_abel”
2. Select its target type : either a literal value (string, date, boolean, integer, etc.), or a resource (URI)
3. Optionally, select some sanitization options, if needed.

Then you need to select the converter to generate the value. The converter to select depends on the use-cases described below.

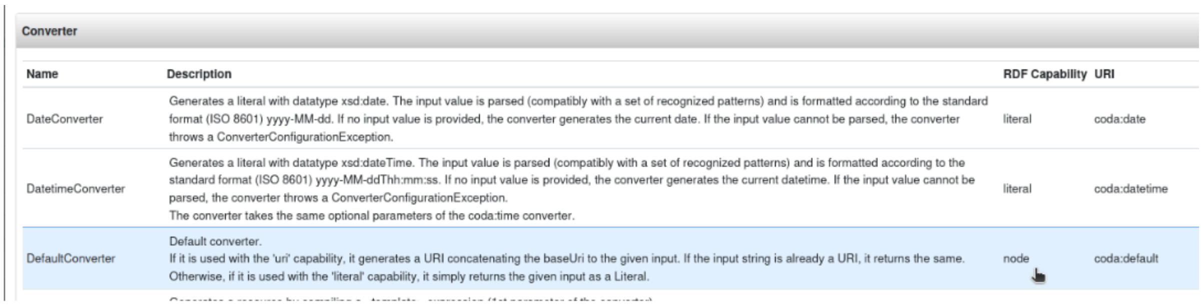
Case 1: map a string column (or with a datatype)

The simplest case is when we have the column contains a basic string value, for example the « label » column in the table below:



	A	B	C	D
1	URI	label	children concepts	related concepts
2	concept-001	A	concept:00006	concept:00007
3	concept-002	B	concept:00007	concept:00008
4	concept-003	C	concept:00008	concept:00009

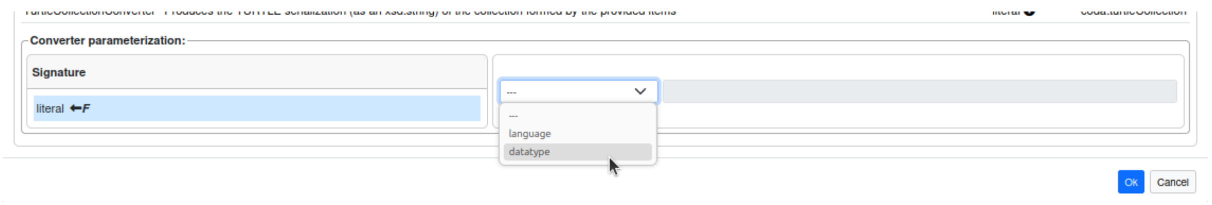
In this case select “**`DefaultConverter`**”.


Converter

Name	Description	RDF Capability	URI
DateConverter	Generates a literal with datatype xsd:date. The input value is parsed (compatibly with a set of recognized patterns) and is formatted according to the standard format (ISO 8601) yyyy-MM-dd. If no input value is provided, the converter generates the current date. If the input value cannot be parsed, the converter throws a ConverterConfigurationException.	literal	coda:date
DatetimeConverter	Generates a literal with datatype xsd:dateTime. The input value is parsed (compatibly with a set of recognized patterns) and is formatted according to the standard format (ISO 8601) yyyy-MM-ddThh:mm:ss. If no input value is provided, the converter generates the current datetime. If the input value cannot be parsed, the converter throws a ConverterConfigurationException. The converter takes the same optional parameters of the coda:time converter.	literal	coda:datetime
DefaultConverter	Default converter. If it is used with the 'uri' capability, it generates a URI concatenating the baseUri to the given input. If the input string is already a URI, it returns the same. Otherwise, if it is used with the 'literal' capability, it simply returns the given input as a Literal.	node	coda:default

The string column will be mapped as is, using the content of the column.

Optionally, you can associate a datatype to the generated value:


Converter parameterization:

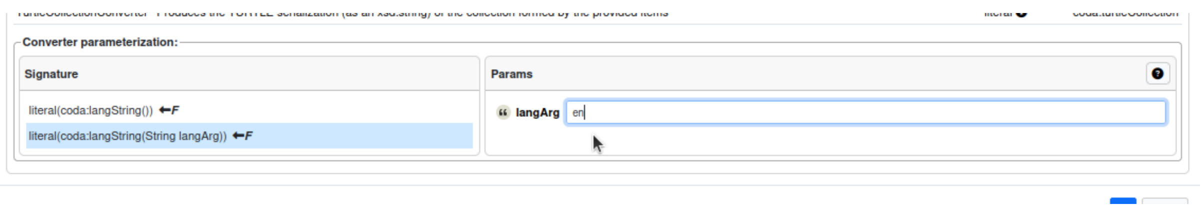
Signature
literal ⬅F

This is useful when your column contains values of a specific pattern, such as booleans "true" / "false", or dates with the structure "yyyy-mm-dd".

Case 2: map a column with a language

In the case where the column contains a string to be associated with a language tag, such as a label, a synonym, or a definition, use "**`LangStringConverter`**".

This converter will generate a literal with a language tag to be specified. Select the signature "`literal(coda:langString(String langArg))`" in the signature panel, and enter the target language code on the right:

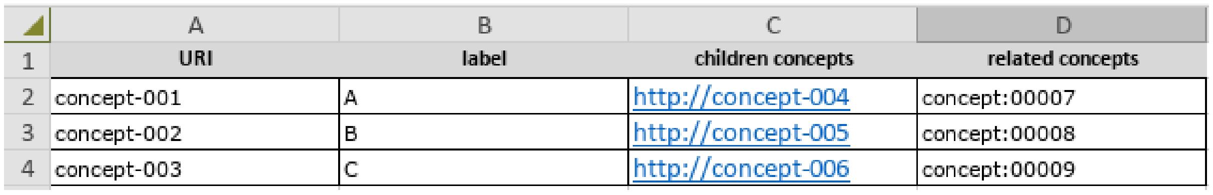


Converter parameterization:

Signature	Params
literal(coda:langString()) ←*F*	langArg: en
literal(coda:langString(String langArg)) ←*F*	

Case 3: map a column containing a complete URI

In the case where the column contains a full URI, like see column C in the example table below:

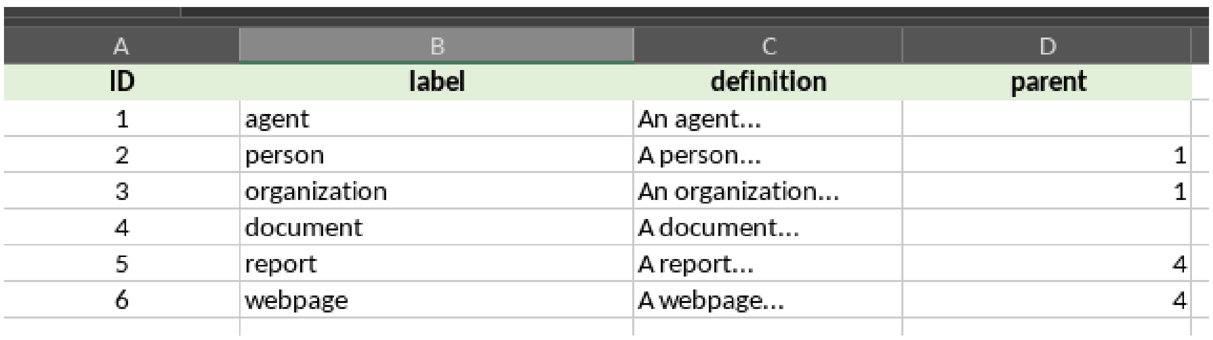


	A	B	C	D
1	URI	label	children concepts	related concepts
2	concept-001	A	http://concept-004	concept:00007
3	concept-002	B	http://concept-005	concept:00008
4	concept-003	C	http://concept-006	concept:00009

Also use "**`DefaultConverter`**". The default converter will automatically determine that the value is a URI, and will use it "as-is", using the content of the column.

Case 4: map a reference to another entry with an ID

There are cases where a column from the table makes a reference to the identifier of another entry, for example to refer to a parent or related entry. This is the case with the "parent" column below:

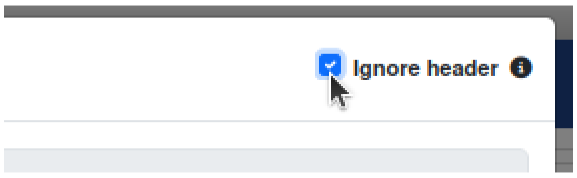


A	B	C	D
ID	label	definition	parent
1	agent	An agent...	
2	person	A person...	1
3	organization	An organization...	1
4	document	A document...	
5	report	A report...	4
6	webpage	A webpage...	4

In this case, use "`DefaultConverter`", which will concatenate the base URI of the project with the content of the column.

Case 5 : explicitly ignoring a column in the mapping

It may happen that columns in the file need to be ignored and will not be converted. In this case, edit this column mapping and tick the "ignore header" checkbox on the top right corner of the "Header editor" dialog:

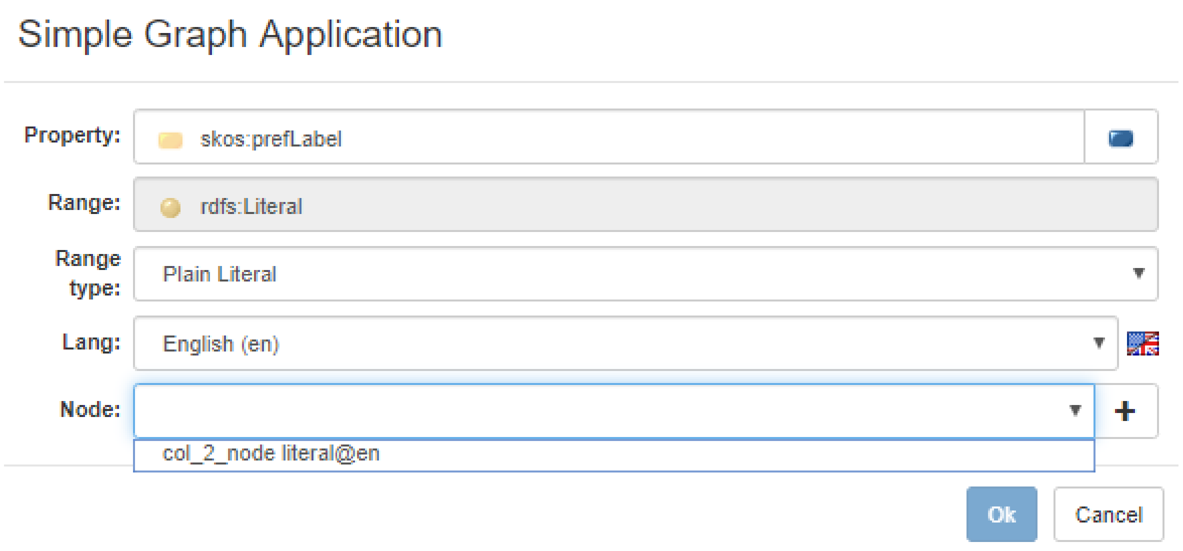


This will make it clear that the column has been reviewed and explicitly excluded from the mapping.

Specify the graph application

Once you have mapped the node, using one the use-cases above, you need to specify how this value will be used in the resulting triple. This is done in the "Graph application" section which allows you to declare an RDF triple model.

In the Graph application section, click + and choose "Simple Graph Application".



Simple Graph Application

Property: skos:prefLabel

Range: rdfs:Literal

Range type: Plain Literal

Lang: English (en)

Node: col_2_node literal@en

Select the target predicate to use in the "Property" field, by clicking on the blue folder. This property must be defined in the ontology underlying your project in VocBench (so the ontology needs to have been imported first).

The Range field is automatically populated and cannot be edited.

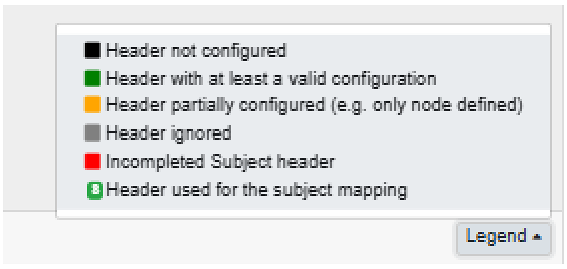
Choose a Range type to specify the type of the triple object. It can only take two values: Literal (for text) or Resource (for a URI). In our example for the definitions, we choose Literal and choose DataType rdf:langString (since we set a language for the Node).

In the “Node” field select the Node you created in the previous section. Most of the time, and for simple mappings, there is only one Node here to select from. This will be the object of the generated triple.

Click OK and then OK again to close the Header editor window.

Status of column mapping : green / orange / red headers

While you work on your column mapping, a legend is available just bottom right of the mapping window : the color of each column header allows you to check the current status of each column mapping.



⬛ Header not configured
🟩 Header with at least a valid configuration
🟧 Header partially configured (e.g. only node defined)
▨ Header ignored
🟥 Incompleted Subject header
🟩 Header used for the subject mapping

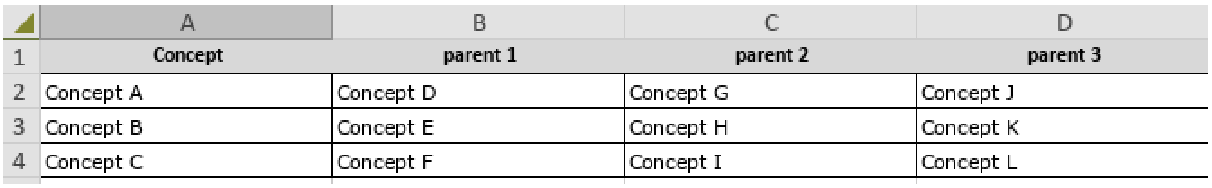
- **Black** : Header not yet configured
- **Green** : Header with at least a valid configuration
- **Orange** : Header partially configured (e.g. only node defined, but not graph application)
- **Gray** : Header was ignored
- **Red** : Incompleted Subject header
- **Green S icon** : Header used for the subject mapping

/!\ Only the green headers will be converted in the end

Wizard : advanced use-cases

How to refer to automatically assigned URI from another column

Consider the example below:

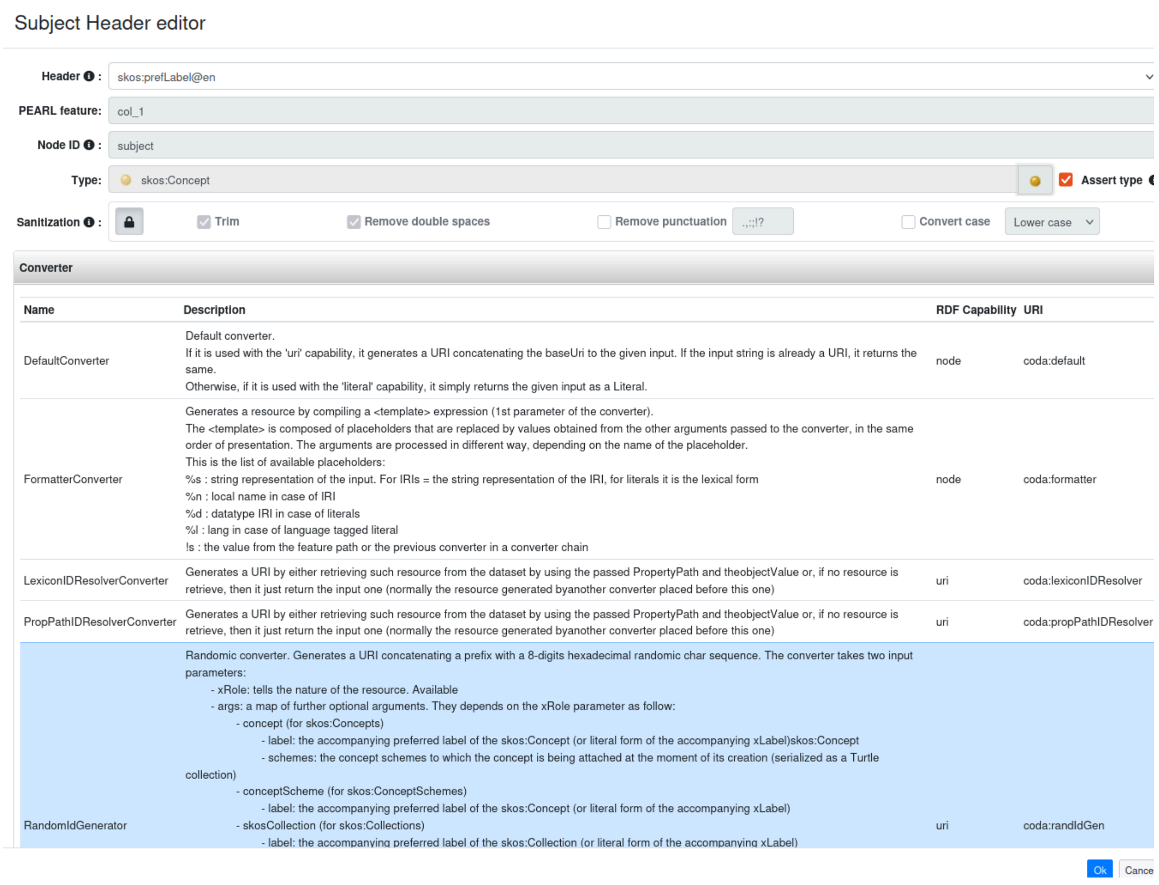


	A	B	C	D
1	**Concept**	**parent 1**	**parent 2**	**parent 3**
2	Concept A	Concept D	Concept G	Concept J
3	Concept B	Concept E	Concept H	Concept K
4	Concept C	Concept F	Concept I	Concept L

The table does not contain any usable ID. URIs will be assigned automatically to each Concept. In the “parent 1”, “parent 2” and “parent 3” columns, we need to refer to the URI that has been automatically assigned to the corresponding Concept.

“Memoize” in the Subject column

When defining the Subject mapping, use the “**`RandomIdGenerator`**”.



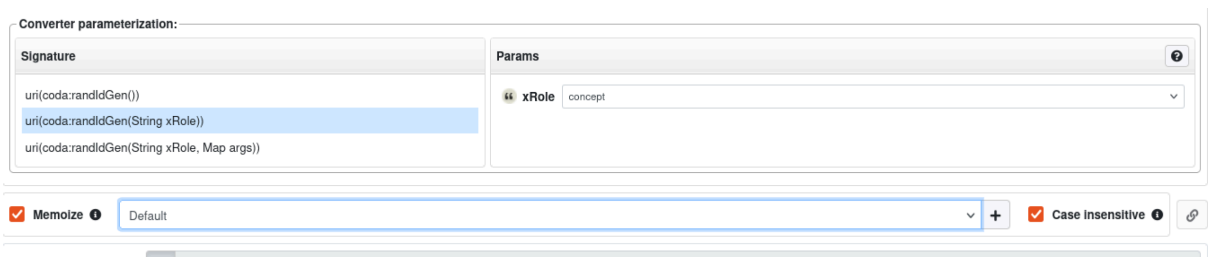
Subject Header editor

Header :	skos:prefLabel@en
PEARL feature:	col_1
Node ID :	subject
Type:	skos:Concept ✓ Assert type
Sanitization :	✓ Trim ✓ Remove double spaces Remove punctuation .,;:!? Convert case Lower case

Converter

Name	Description	RDF Capability	URI
DefaultConverter	Default converter. If it is used with the 'uri' capability, it generates a URI concatenating the baseUri to the given input. If the input string is already a URI, it returns the same. Otherwise, if it is used with the 'literal' capability, it simply returns the given input as a Literal.	node	coda:default
FormatterConverter	Generates a resource by compiling a <template> expression (1st parameter of the converter). The <template> is composed of placeholders that are replaced by values obtained from the other arguments passed to the converter, in the same order of presentation. The arguments are processed in different way, depending on the name of the placeholder. This is the list of available placeholders: %s : string representation of the input. For IRIs = the string representation of the IRI, for literals it is the lexical form %n : local name in case of IRI %d : datatype IRI in case of literals %l : lang in case of language tagged literal !s : the value from the feature path or the previous converter in a converter chain	node	coda:formatter
LexiconIDResolverConverter	Generates a URI by either retrieving such resource from the dataset by using the passed PropertyPath and theobjectValue or, if no resource is retrieve, then it just return the input one (normally the resource generated byanother converter placed before this one)	uri	coda:lexiconIDResolver
PropPathIDResolverConverter	Generates a URI by either retrieving such resource from the dataset by using the passed PropertyPath and theobjectValue or, if no resource is retrieve, then it just return the input one (normally the resource generated byanother converter placed before this one)	uri	coda:propPathIDResolver
RandomIdGenerator	Randomic converter. Generates a URI concatenating a prefix with a 8-digits hexadecimal randomic char sequence. The converter takes two input parameters: - xRole: tells the nature of the resource. Available - args: a map of further optional arguments. They depends on the xRole parameter as follow: - concept (for skos:Concepts) - label: the accompanying preferred label of the skos:Concept (or literal form of the accompanying xLabel)skos:Concept - schemes: the concept schemes to which the concept is being attached at the moment of its creation (serialized as a Turtle collection) - conceptScheme (for skos:ConceptSchemes) - label: the accompanying preferred label of the skos:Concept (or literal form of the accompanying xLabel) - skosCollection (for skos:Collections) - label: the accompanying preferred label of the skos:Collection (or literal form of the accompanying xLabel)	uri	coda:randIdGen

Use its second signature, with a “role”, and select role “concept” (because we are importing concepts)



Converter parameterization:

Signature	Params
uri(coda:randIdGen())	xRole: concept
uri(coda:randIdGen(String xRole))	
uri(coda:randIdGen(String xRole, Map args))	

✓ Memoize: Default ✓ Case insensitive

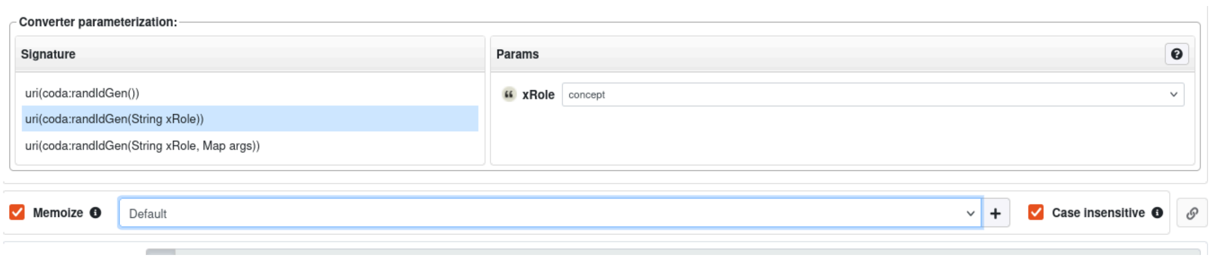
Then check the “Memoize” option. Because we know we will need to refer to those same URIs later (in the parent column), this will create a “cache” of all generated URIs associated to the content of the column (= the label), so that we can refer to this cache later.

You can manage multiple caches with the cache name. In simple cases, leave the “Default” cache, but optionally you can give a name to the cache by clicking the “+” (if we have multiple URI assignment to do in multiple columns, but this is not our case here)

“Memoize” in the column mappings

When mapping each of the “parent 1”, “parent 2” and “parent 3” columns, use the same procedure:

- Use the “**`RandomIdGenerator`**”.
- Use its second signature, with a “role”, and select role “concept” (because we are importing concepts)
- Check the “Memoize” option
- Select the same cache name as the one you picked up during Subject mapping (“Default” by default)


Converter parameterization:

Signature	Params
uri(coda:randIdGen())	xRole: concept
uri(coda:randIdGen(String xRole))	
uri(coda:randIdGen(String xRole, Map args))	

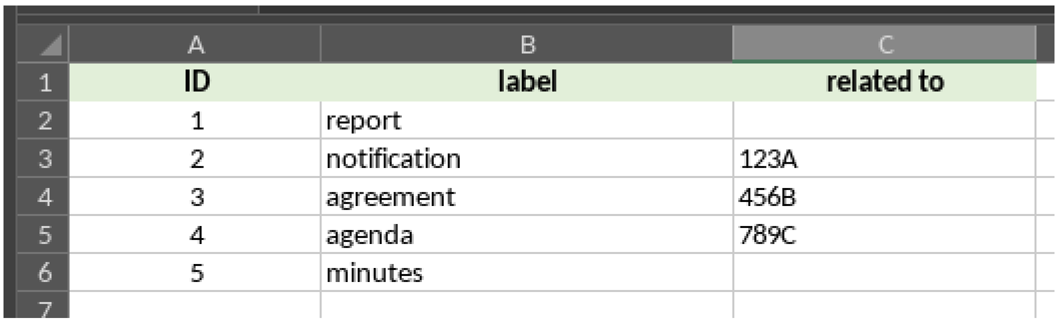
[✓] Memoize — Default — [✓] Case insensitive

This tells Sheet2RDF : *“Whenever you find a label to which you already assigned a URI before (in the cache) then reuse that same URI”*

/!\ Warning : if the column contains a label that was not part of the Subject mapping, then a new URI will be created. The consistency of the table needs to be ensured before importing !

How to lookup URI of existing entries in VocBench from their label

Consider the following example:



	A	B	C
1	ID	label	related to
2	1	report	
3	2	notification	123A
4	3	agreement	456B
5	4	agenda	789C
6	5	minutes	
7			

In this table, the “related to” column contains a reference to some ID of an existing Concept already imported in VocBench. This is not the URI, but could be e.g. the value of a `skos:notation` or `dct:identifier` property associated with the Concept. We would like this column to be mapped to a `skos:relatedMatch` triple.

We will need to do a “lookup” in VocBench to find the URI of the corresponding Concept with a “**`PropPathIDResolver`**”.

To proceed we need to create 2 Nodes (= 2 values) instead of just one:

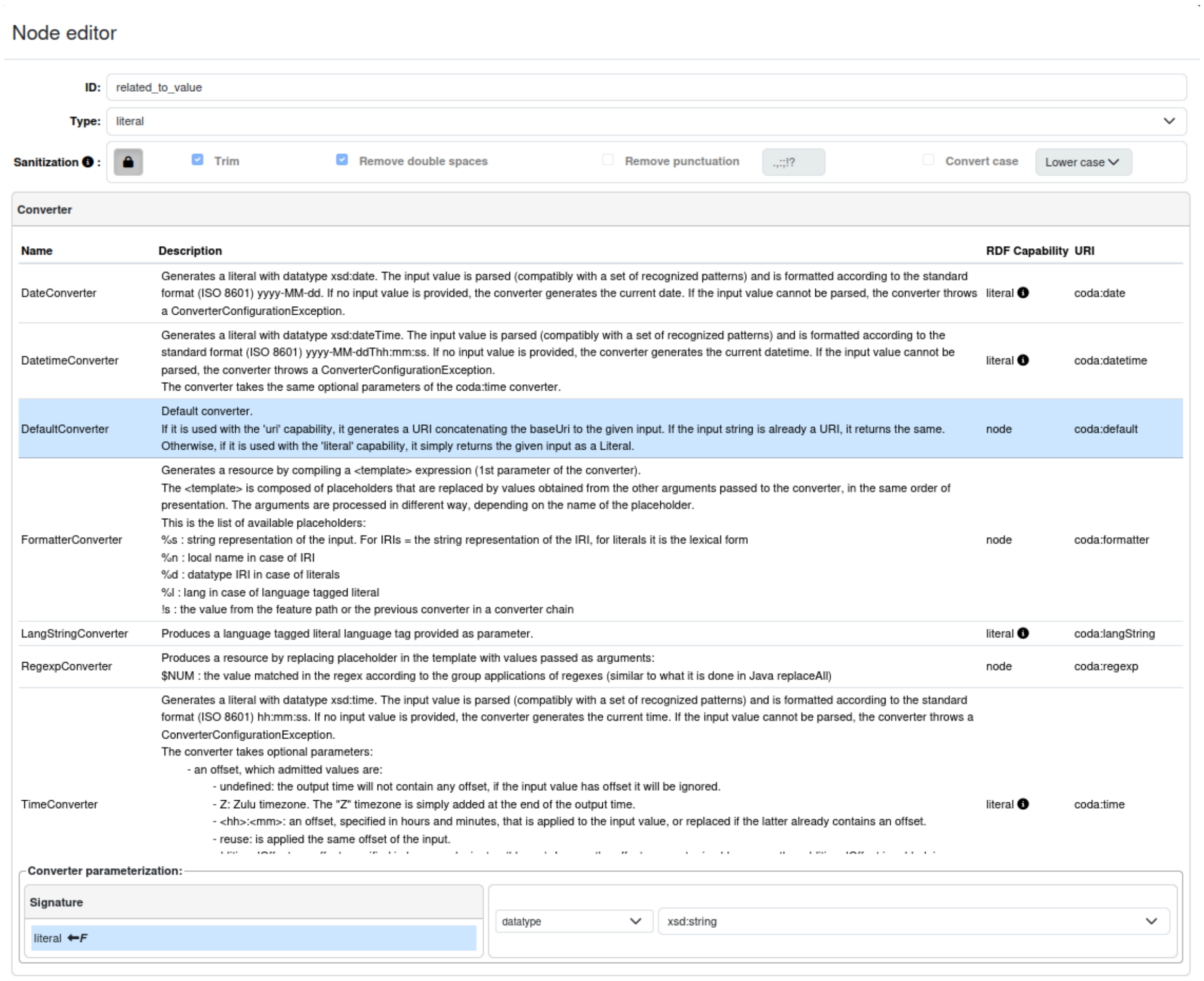
- first, one node that will be the formal value used to search in VocBench with proper language or datatype
- second node that will be the result of searching in VocBench the URI that has the first node as the value of a given property

/!\ Order matters here ! the nodes need to be created in this order.

Define the first node containing the exact value to lookup

First create a first node that maps the column value to the exact value to be searched in VocBench. By “exact value” we mean the literal with the adequate datatype or language tag. To do this, follow the steps explained above and use either a “**`DefaultConverter`**” or a “**`LangStringConverter`**”.

In our example we create a node called “related_to_value”, of type Literal, with a DefaultConverter, and an xsd:string datatype because we know that the value to lookup have this structure:



Node editor

ID:	related_to_value
Type:	literal
Sanitization:	Trim [✓] · Remove double spaces [✓] · Remove punctuation [ ] .,:;!? · Convert case [ ] Lower case

Converter

Name	Description	RDF Capability	URI
DateConverter	Generates a literal with datatype xsd:date. The input value is parsed (compatibly with a set of recognized patterns) and is formatted according to the standard format (ISO 8601) yyyy-MM-dd. If no input value is provided, the converter generates the current date. If the input value cannot be parsed, the converter throws a ConverterConfigurationException.	literal	coda:date
DatetimeConverter	Generates a literal with datatype xsd:dateTime. The input value is parsed (compatibly with a set of recognized patterns) and is formatted according to the standard format (ISO 8601) yyyy-MM-ddThh:mm:ss. If no input value is provided, the converter generates the current datetime. If the input value cannot be parsed, the converter throws a ConverterConfigurationException. The converter takes the same optional parameters of the coda:time converter.	literal	coda:datetime
DefaultConverter	Default converter. If it is used with the 'uri' capability, it generates a URI concatenating the baseUri to the given input. If the input string is already a URI, it returns the same. Otherwise, if it is used with the 'literal' capability, it simply returns the given input as a Literal.	node	coda:default
FormatterConverter	Generates a resource by compiling a <template> expression (1st parameter of the converter). The <template> is composed of placeholders that are replaced by values obtained from the other arguments passed to the converter, in the same order of presentation. The arguments are processed in different way, depending on the name of the placeholder. This is the list of available placeholders: %s : string representation of the input. For IRIs = the string representation of the IRI, for literals it is the lexical form %n : local name in case of IRI %d : datatype IRI in case of literals %l : lang in case of language tagged literal !s : the value from the feature path or the previous converter in a converter chain	node	coda:formatter
LangStringConverter	Produces a language tagged literal language tag provided as parameter.	literal	coda:langString
RegexpConverter	Produces a resource by replacing placeholder in the template with values passed as arguments: $NUM : the value matched in the regex according to the group applications of regexes (similar to what it is done in Java replaceAll)	node	coda:regexp
TimeConverter	Generates a literal with datatype xsd:time. The input value is parsed (compatibly with a set of recognized patterns) and is formatted according to the standard format (ISO 8601) hh:mm:ss. If no input value is provided, the converter generates the current time. If the input value cannot be parsed, the converter throws a ConverterConfigurationException. The converter takes optional parameters: - an offset, which admitted values are: - undefined: the output time will not contain any offset, if the input value has offset it will be ignored. - Z: Zulu timezone. The "Z" timezone is simply added at the end of the output time. - <hh>:<mm>: an offset, specified in hours and minutes, that is applied to the input value, or replaced if the latter already contains an offset. - reuse: is applied the same offset of the input.	literal	coda:time

Converter parameterization:

Signature
literal ⬅*F*

datatype: xsd:string

After the node creation don't proceed to the "Graph application" yet ! as this first node is only used to do the lookup, but will not be mapped directly to the graph.

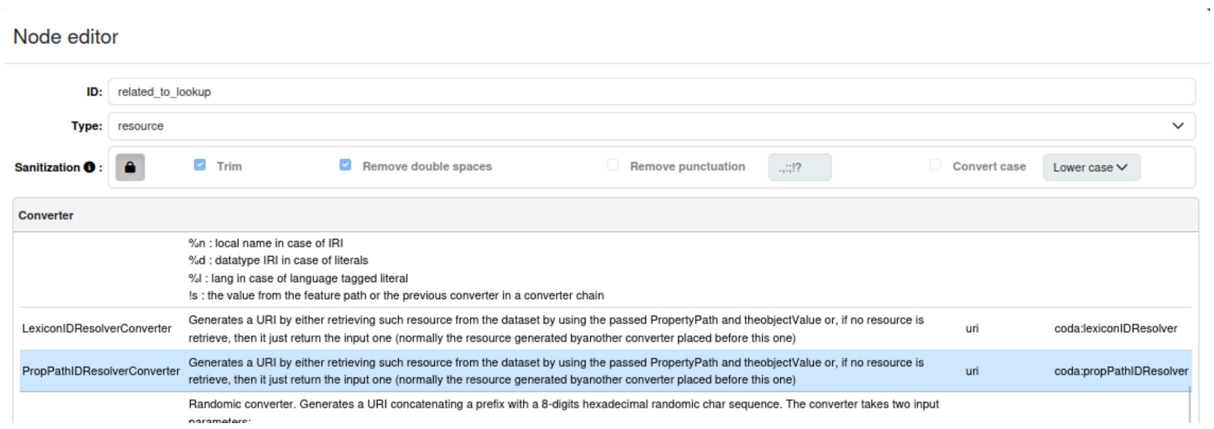
Define the second node containing the lookup result

Now declare another node. When declaring the mapping for this node, we will do a lookup of the first node.

To do this you can either use:

- The "**`PropPathIDResolverConverter`**" that searches the value in a property to be specified (e.g. `skos:notation`)
- The "**`LexiconIDResolverConverter`**" that searches for labels (prefLabel, altLabel or other kinds of labels, depending on the lexicalisation parameter of the project).

We will use the "**`PropPathIDResolverConverter`**". Our second node is called "related_to_lookup", and it is of type "Resource" because it will contain the URI of the Resource having the specified ID.



Node editor

ID: related_to_lookup

Type: resource

Sanitization: Trim, Remove double spaces, Remove punctuation .,;:!?, Convert case, Lower case

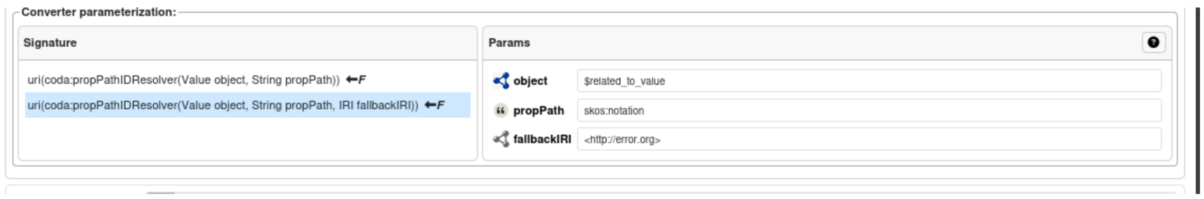
Converter

	%n : local name in case of IRI %d : datatype IRI in case of literals %l : lang in case of language tagged literal !s : the value from the feature path or the previous converter in a converter chain		
LexiconIDResolverConverter	Generates a URI by either retrieving such resource from the dataset by using the passed PropertyPath and theobjectValue or, if no resource is retrieve, then it just return the input one (normally the resource generated byanother converter placed before this one)	uri	coda:lexiconIDResolver
PropPathIDResolverConverter	Generates a URI by either retrieving such resource from the dataset by using the passed PropertyPath and theobjectValue or, if no resource is retrieve, then it just return the input one (normally the resource generated byanother converter placed before this one)	uri	coda:propPathIDResolver
	Randomic converter. Generates a URI concatenating a prefix with a 8-digits hexadecimal randomic char sequence. The converter takes two input parameters:		

In the Signature panel we use the signature with 3 parameters:

1. The value that we search : this is a reference to the first node we created in the previous step, preceded by a "$" sign, in our case `$related_to_value`
2. The full URI of the property that we search. This can be a prefixed URI if the prefix is known, in our case `skos:notation`
3. A fallback value if not found : "`<http://error.org>`" - so that we can check after conversion that this fallback value is never present. Use angle brackets around this URI.

The parameters look like this:



Converter parameterization:

Signature

uri(coda:propPathIDResolver(Value object, String propPath)) ⇐F

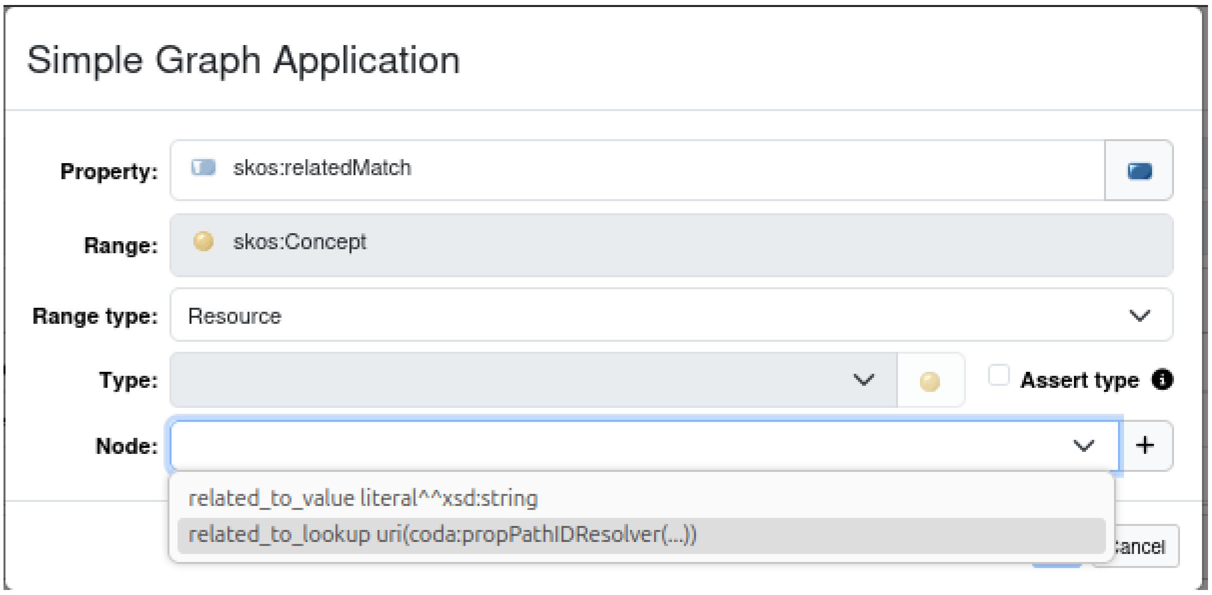
uri(coda:propPathIDResolver(Value object, String propPath, IRI fallbackIRI)) ⇐F

Params

object	$related_to_value
propPath	skos:notation
fallbackIRI	<http://error.org>

Create the graph application

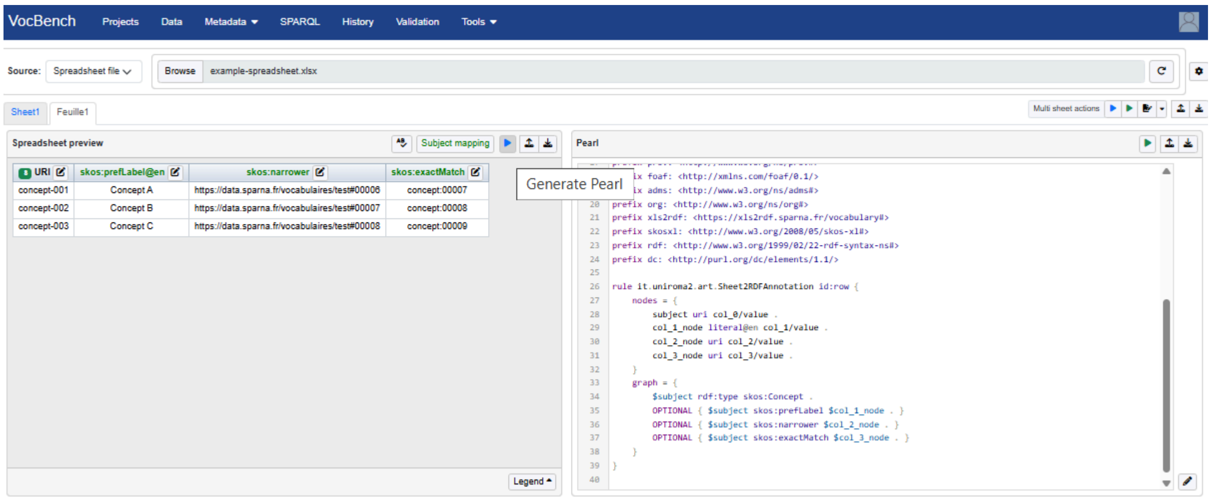
When defining the Simple Graph Application, select the desired predicate as usual (`skos:relatedMatch` in our case) and make sure you select the second node (`related_to_lookup` in our case) :



Wizard conclusion : Generate the PEARL mapping

When the mapping of every column is done, click the blue “Play” arrow in the top right of the mapping panel to generate the mapping in the PEARL code panel.

The Pearl section shows the Pearl code generated from the configurations you have made. You do not need to know the Pearl programming language to use the tool.



URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
concept-001	Concept A	https://data.sparna.fr/vocabulaires/test#00006	concept:00007
concept-002	Concept B	https://data.sparna.fr/vocabulaires/test#00007	concept:00008
concept-003	Concept C	https://data.sparna.fr/vocabulaires/test#00008	concept:00009

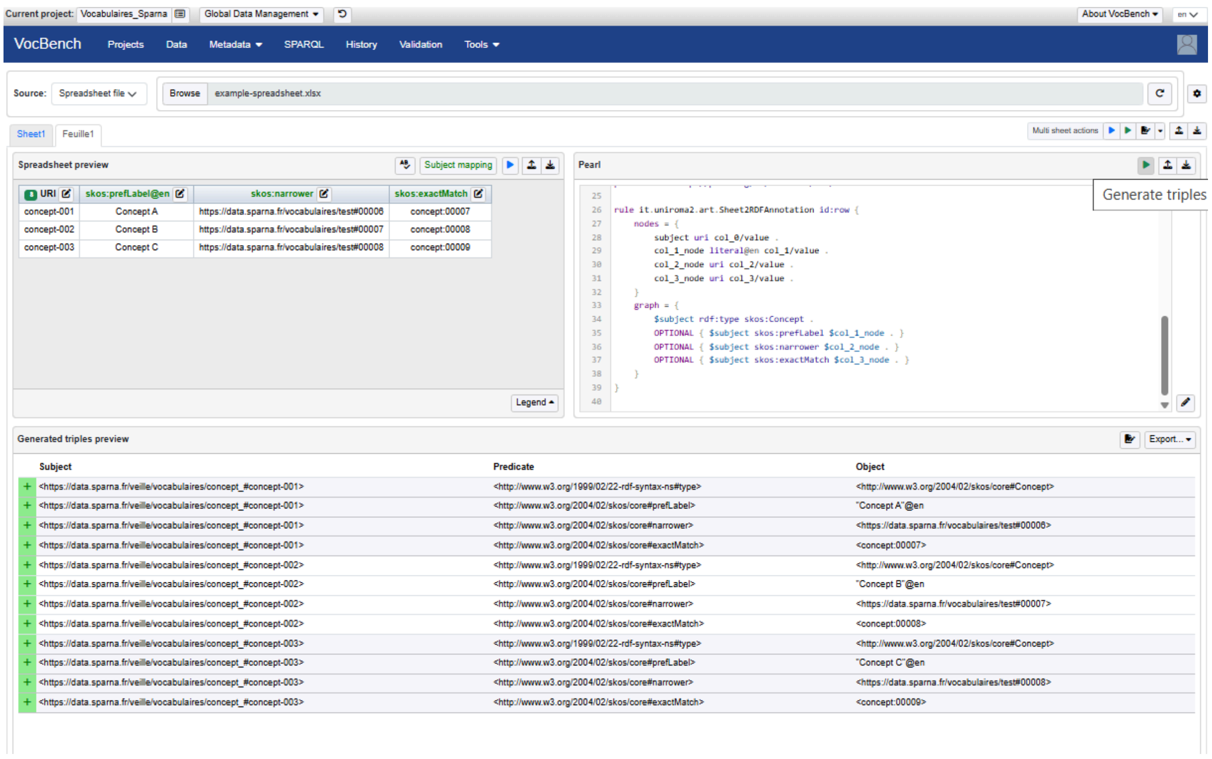
```
ix foaf: <http://xmlns.com/foaf/0.1/>
ix adms: <http://www.w3.org/ns/adms#>
prefix org: <http://www.w3.org/ns/org#>
prefix xls2rdf: <https://xls2rdf.sparna.fr/vocabulary#>
prefix skosxl: <http://www.w3.org/2008/05/skos-xl#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix dc: <http://purl.org/dc/elements/1.1/>

rule it.uniroma2.art.Sheet2RDFAnnotation id:row {
    nodes = {
        subject uri col_0/value .
        col_1_node literal@en col_1/value .
        col_2_node uri col_2/value .
        col_3_node uri col_3/value .
    }
    graph = {
        $subject rdf:type skos:Concept .
        OPTIONAL { $subject skos:prefLabel $col_1_node . }
        OPTIONAL { $subject skos:narrower $col_2_node . }
        OPTIONAL { $subject skos:exactMatch $col_3_node . }
    }
}
```

Optional: it is possible to modify PEARL manually (to create conditional rules) - see the advanced section for cases where you may want to modify the mapping manually.

For more information, see the [Pearl documentation.]()

Execute the PEARL mapping

Once the PEARL mapping is generated, you can apply it to convert the spreadsheet. Click the Green Play button in the Pearl window to generate the triplets. You can explore the triples in the Generated Triples Preview window at the bottom.



Source: Spreadsheet file — example-spreadsheet.xlsx

Spreadsheet preview

URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
concept-001	Concept A	https://data.sparna.fr/vocabulaires/test#00006	concept:00007
concept-002	Concept B	https://data.sparna.fr/vocabulaires/test#00007	concept:00008
concept-003	Concept C	https://data.sparna.fr/vocabulaires/test#00008	concept:00009

Pearl

```
rule it.uniroma2.art.Sheet2RDFAnnotation id:row {
    nodes = {
        subject uri col_0/value .
        col_1_node literal@en col_1/value .
        col_2_node uri col_2/value .
        col_3_node uri col_3/value .
    }
    graph = {
        $subject rdf:type skos:Concept .
        OPTIONAL { $subject skos:prefLabel $col_1_node . }
        OPTIONAL { $subject skos:narrower $col_2_node . }
        OPTIONAL { $subject skos:exactMatch $col_3_node . }
    }
}
```

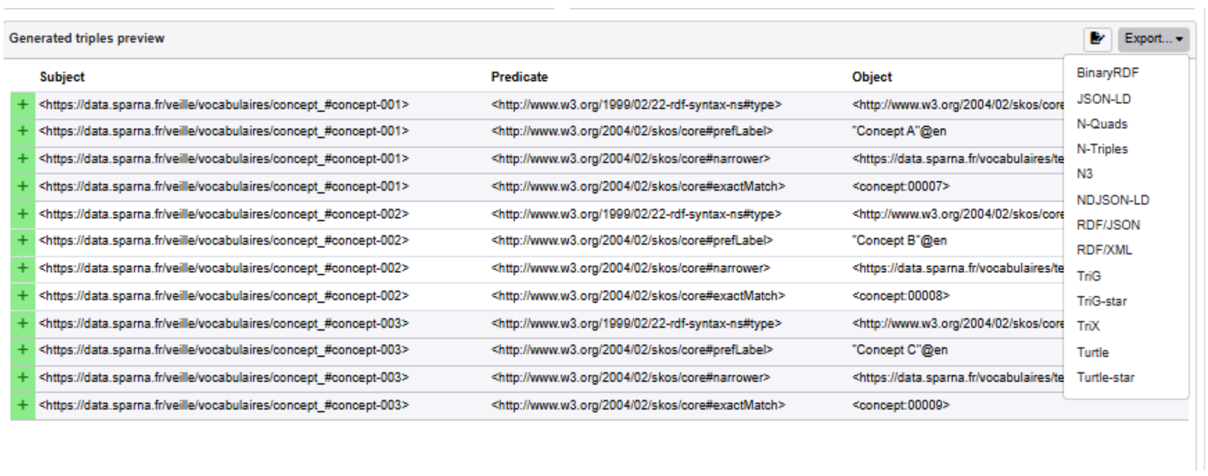
Generated triples preview

	Subject	Predicate	Object
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core#Concept>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"Concept A"@en
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/2004/02/skos/core#narrower>	<https://data.sparna.fr/vocabulaires/test#00006>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/2004/02/skos/core#exactMatch>	<concept:00007>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core#Concept>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"Concept B"@en
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/2004/02/skos/core#narrower>	<https://data.sparna.fr/vocabulaires/test#00007>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/2004/02/skos/core#exactMatch>	<concept:00008>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core#Concept>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"Concept C"@en
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/2004/02/skos/core#narrower>	<https://data.sparna.fr/vocabulaires/test#00008>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/2004/02/skos/core#exactMatch>	<concept:00009>

Download or import the resulting triples

The last section, Generated triples preview, allows you to check the result of the transformation into RDF.

Here you can export the generated triples under various formats :



	Subject	Predicate	Object
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"Concept A"@en
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/2004/02/skos/core#narrower>	<https://data.sparna.fr/vocabulaires/te
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-001>	<http://www.w3.org/2004/02/skos/core#exactMatch>	<concept:00007>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"Concept B"@en
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/2004/02/skos/core#narrower>	<https://data.sparna.fr/vocabulaires/te
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-002>	<http://www.w3.org/2004/02/skos/core#exactMatch>	<concept:00008>
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2004/02/skos/core
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/2004/02/skos/core#prefLabel>	"Concept C"@en
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/2004/02/skos/core#narrower>	<https://data.sparna.fr/vocabulaires/te
+	<https://data.sparna.fr/veille/vocabulaires/concept_#concept-003>	<http://www.w3.org/2004/02/skos/core#exactMatch>	<concept:00009>

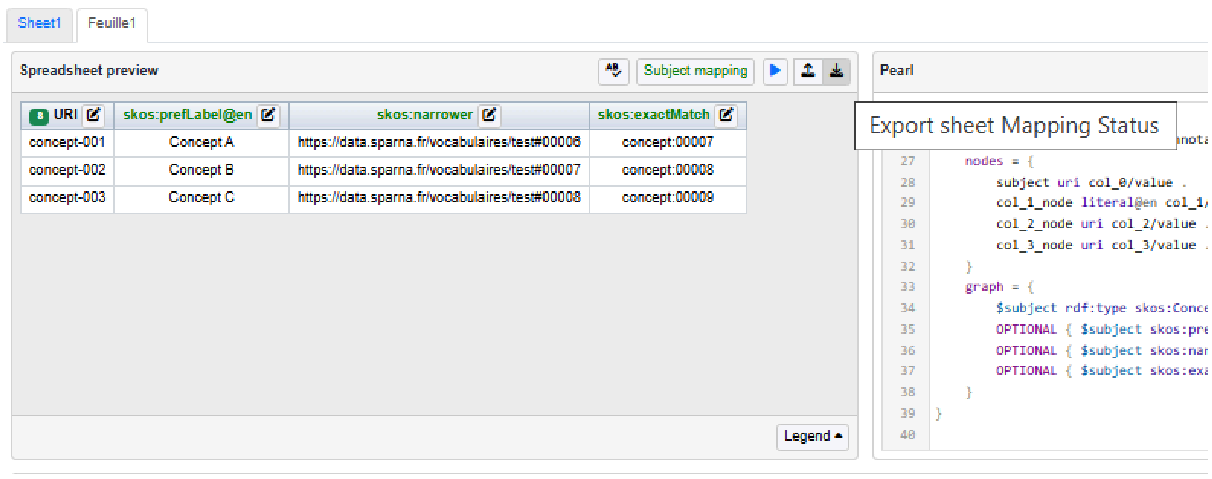
If you prefer to use VocBench to work on your resource, click on the “Add triples” button to populate your RDF database. The triples then get inserted inside the VocBench project.

Load / save the mapping wizard or the PEARL mapping

Both the mapping configuration and the PEARL mapping can be saved and reloaded.

Load / save the mapping wizard

The configuration template can be exported using the download buttons and used to reproduce this process identically and effortlessly on another similar dataset.

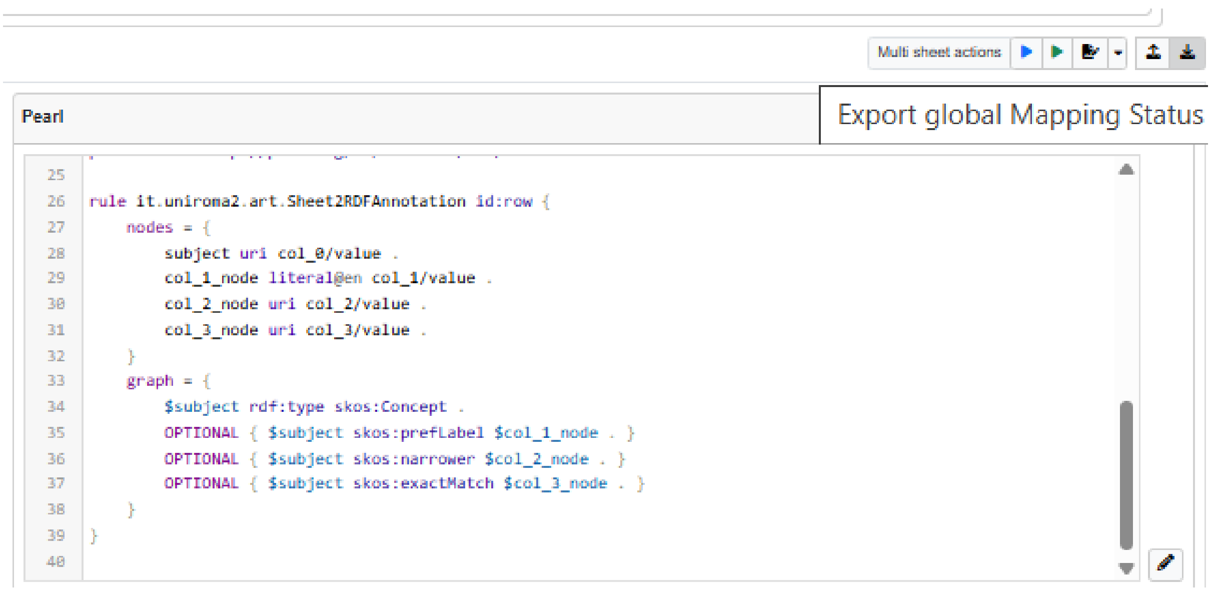


URI	skos:prefLabel@en	skos:narrower	skos:exactMatch
concept-001	Concept A	https://data.sparna.fr/vocabulaires/test#00006	concept:00007
concept-002	Concept B	https://data.sparna.fr/vocabulaires/test#00007	concept:00008
concept-003	Concept C	https://data.sparna.fr/vocabulaires/test#00008	concept:00009

```
                                                    nota
    nodes = {
        subject uri col_0/value .
        col_1_node literal@en col_1/
        col_2_node uri col_2/value .
        col_3_node uri col_3/value .
    }
    graph = {
        $subject rdf:type skos:Conce
        OPTIONAL { $subject skos:pre
        OPTIONAL { $subject skos:nar
        OPTIONAL { $subject skos:exa
    }
}
```

Load / save the PEARL mapping

The generated Pearl script can be exported using the download buttons and used to reproduce this process identically and effortlessly on another similar dataset

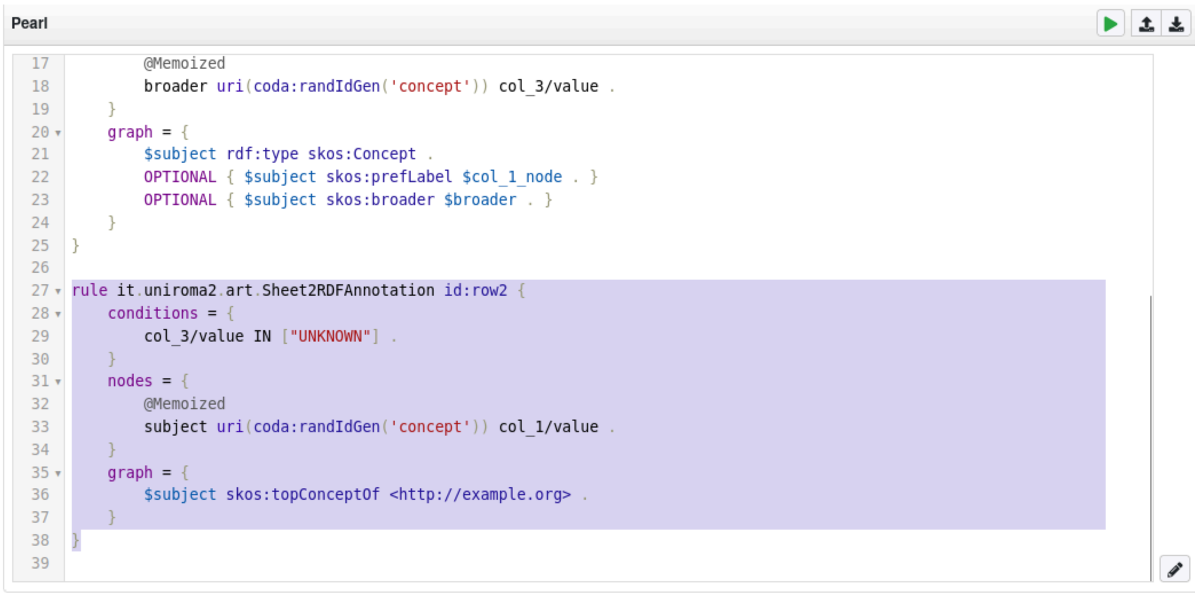


```
rule it.uniroma2.art.Sheet2RDFAnnotation id:row {
    nodes = {
        subject uri col_0/value .
        col_1_node literal@en col_1/value .
        col_2_node uri col_2/value .
        col_3_node uri col_3/value .
    }
    graph = {
        $subject rdf:type skos:Concept .
        OPTIONAL { $subject skos:prefLabel $col_1_node . }
        OPTIONAL { $subject skos:narrower $col_2_node . }
        OPTIONAL { $subject skos:exactMatch $col_3_node . }
    }
}
```

Advanced : manual PEARL mapping edition (conditional rules)

Some mapping rules need to be manually done by directly editing the PEARL mapping and cannot be defined in the mapping wizard. This is the case for conditional rules, when we need to generate triples only in the case the value in the column is equal, or different from, a certain value.
Take for example the case of the skos:topConceptOf predicate to relate the root concepts of a taxonomy to their ConceptScheme: this predicate has to be created only for root concepts, not for others.
The PEARL language documentation contains a dedicated section on conditional rules.

Conditions are expressed in an additional `conditions { … }` section in the rule. The example below illustrates how the skos:topConceptOf predicate is generated only if the third column (being the column indicating the parent of the concept) contains the value “UNKNOWN”.



```
        @Memoized
        broader uri(coda:randIdGen('concept')) col_3/value .
    }
    graph = {
        $subject rdf:type skos:Concept .
        OPTIONAL { $subject skos:prefLabel $col_1_node . }
        OPTIONAL { $subject skos:broader $broader . }
    }
}

rule it.uniroma2.art.Sheet2RDFAnnotation id:row2 {
    conditions = {
        col_3/value IN ["UNKNOWN"] .
    }
    nodes = {
        @Memoized
        subject uri(coda:randIdGen('concept')) col_1/value .
    }
    graph = {
        $subject skos:topConceptOf <http://example.org> .
    }
}
```

The rule will be triggered only when the condition is set. Note that it is not possible to test for an empty value in the table (as empty cells do not generate a corresponding value in the mapping process).