How-to configure Sparnatural

date: 2023-10-20
Sparnatural version: 8.5.0

Introduction

Welcome to this guide on how to configure Sparnatural !

The Sparnatural OWL configuration reference documentation lists the available annotations and axioms available to configure Sparnatural. In this documentation you will learn how to use these annotations concretely and define the classes, properties, widgets and datasources in order to make your Sparnatural explorer as appealing as possible for your users.

Conventions

URIs are indicated like this.
Headers in the spreadsheet are indicated like this.

 **Important** : this is an important note, pay attention !

 **Advanced note**: this is explaining something advanced. Don't worry if you don't understand all the details at first.

 **Tip**: this is a useful and practical tip.

Prerequisites

1. Make sure you have followed the introductory "Hello Sparnatural" guide to setup your environment to point Sparnatural to your triplestore and adjust the browser security settings.
2. You must have a local spreadsheet editor, like Microsoft Excel.
3. You need to have a basic understanding of OWL ontologies.
4. For configuring your own datasource queries, you need to be proficient with SPARQL. This is described in annex.

In addition, you can have the Protégé OWL editor installed, only if you want to browse the ontology in Protégé, but this is not a requirement.

Documentation files

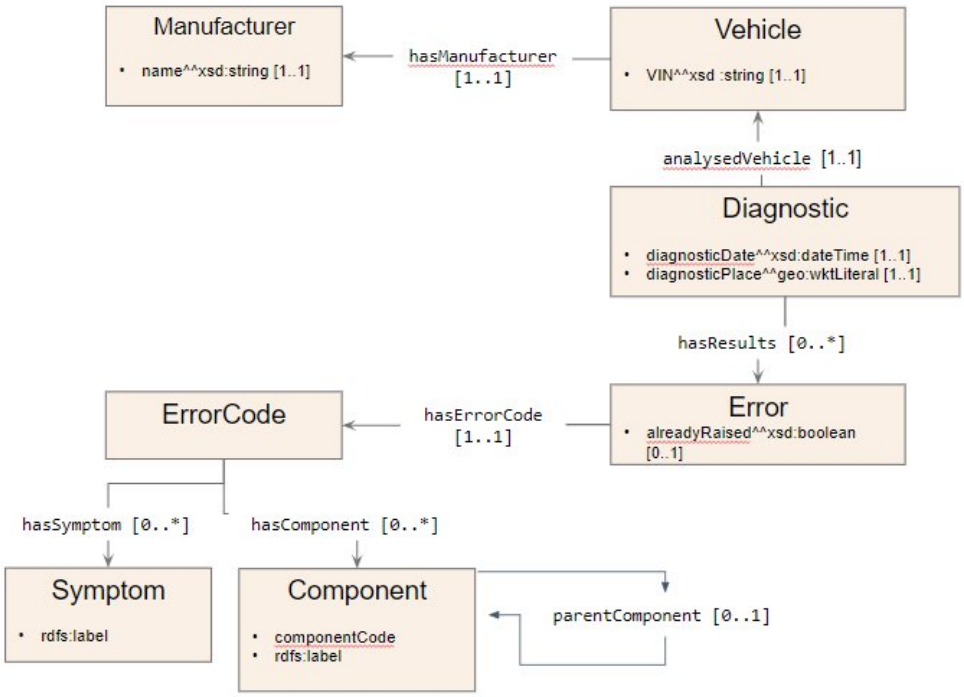
This guide comes with a set of files that you should have ready. Click on the links to download them:

1. **car.ttl** : a sample OWL ontology describing car diagnostics.

2. **car_instances.ttl** : a few manually crafted instances of the sample ontology. Although not strictly required, you should load these instances into your triplestore if you want to follow along and test the example configuration against the dataset.
3. **sparnatural-car-configuration.xlsx** : the example Sparnatural Excel config file
4. **sparnatural-car-configuration.ttl** : the result of converting the Excel config file with the Excel-2-RDF converter. This is the actual Sparnatural configuration file to pass in the “src” attribute of the sparnatural HTML element, if you want to test it to see the final result (but this is not required to follow this documentation).

Structure of the example ontology

For the purpose of this documentation we will use an example ontology, defined in car.ttl, and described in the following diagram:



This is a simplistic representation of “On-board diagnostic” systems of cars : Vehicles, identified by their Vehicle Identification Number (VIN) have a manufacturer; Diagnostics are made on given vehicles at a certain date and a certain place, and can yield errors. An error has a code, and a flag indicating if the error was already detected on the same vehicle. Error codes are associated with symptoms (“Engine Misfire” or "Transmission Slipping") and components (“Engine”, “Transmission”, “Brakes”). Components are hierarchically organized. The ontology uses the prefix “odb” associated with the URI http://example.com/ontology/odb#.

Disclaimer : this “car” ontology sample is a fictitious one, which has only been created for the purpose of testing maximum Sparnatural different functionalities. This ontology might not be fully exact nor complete in a real car diagnostic industrial context ! [1]

[1] Other Sparnatural beginners happen to test the tool with cultural or library metadata, small foaf (“friend

Configuration spreadsheet

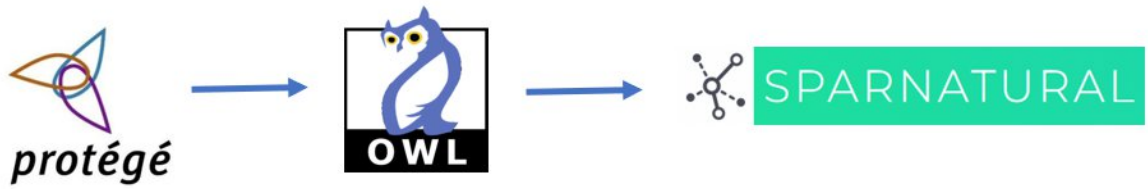
Protégé vs. spreadsheet

Sparnatural can be configured by an OWL ontology, and the “Hello Sparnatural” guide explains how to use the Protégé OWL editor to start creating an OWL config ontology for Sparnatural.

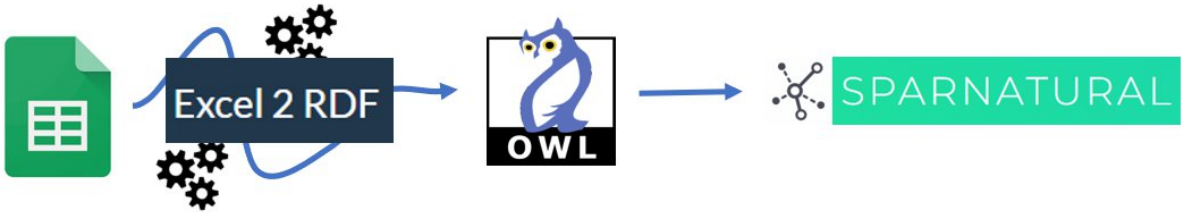
Although configuration in Protégé offers navigation and edition UI in trees of classes and properties, and although semantic web practitioners are familiar with it, we must admit it ain’t the fastest editing solution 😊

No worries then ! Spreadsheet configuration we present in this document is faster and easier to go and the result is the same : an OWL file that Sparnatural can read. The config can even be edited live in case of online spreadsheets !

When using Protégé, you directly edit an OWL file:



The conversion of the spreadsheet into OWL relies on a generic Excel-to-RDF converter. While when using a spreadsheet, the Excel-2-RDF converter is used:



---

of a friend”) structures using “knows” or other properties to develop a random mini-knowledge graph. Some even invented something with pets (owners, names, homes and sounds !), or relied on the Stanford’s fictitious pizza ontology for Protégé demo.

The Excel-2-RDF converter

The code of the converter is open-sourced in the xls2rdf Github repository. The Excel-2-RDF converter is available in different packagings:

1. an online REST service
2. an online form where you can upload your file
3. a command-line converter with its documentation
4. a Java library file to be integrated into your application

All these “packagings” behave the same way for the conversion of the spreadsheet in RDF. For the purpose of following this documentation, we suggest either using an online Google spreadsheet and rely on the online conversion service, or simply use a local file and upload it through the online form, and save the resulting OWL file.

The detailed behavior of the Excel-to-RDF converter as to how the Excel file is interpreted is out of scope of this guide, and is documented in the online converter service.

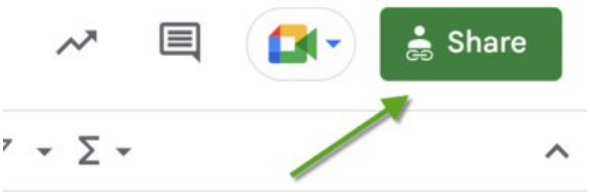
If you use a Google spreadsheet

Using a Google spreadsheet has the following advantages:

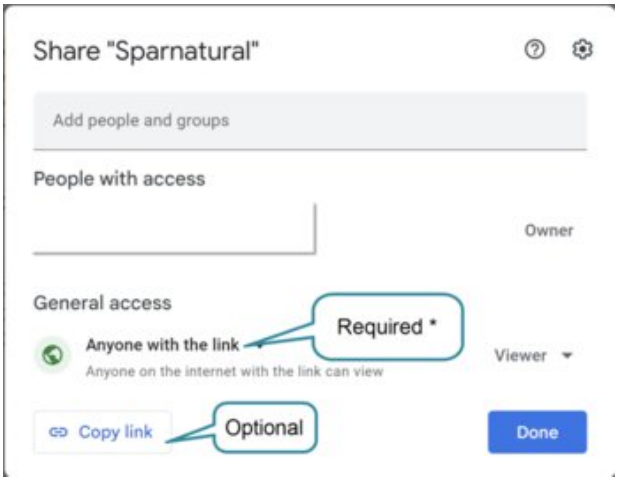
1. The configuration is “live” : while in the test phase, you can edit your spreadsheet, refresh your Sparnatural HTML page, and it will be updated automatically.
2. Multiple persons can collaborate on the same config spreadsheet.

To initialize your configuration spreadsheet:

1. Make a copy of the configuration template
2. Your spreadsheet needs to be publicly visible. You need to share it with the *"Anyone with the link = Viewer"* option. To do this, select the option **Share**.



In the next window, click the **"General access"** button. Select the *"Anyone with the link"* option and press the "Done" button.



After you close the window, copy the URL of the spreadsheet in your browser's address bar.

3. Copy this URL in the cell B2 of the configuration file. Make sure the URL does not end with "/edit#gid=xxxxxxx", remove this part of the URL manually. The URL should look like *https://docs.google.com/spreadsheets/d/xxxxxxxxxx*"
4. Save the content of cell B3 (in red) : this is the configuration URL that you can pass to the "src" attribute of the <spar-natural> HTML element. You see it starts with https://xls2rdf.sparna.fr : this is the online Excel-2-RDF conversion service that takes the Google spreadsheet URL as a parameter. Each time your sparnatural page will load, it will call this URL of the converter, which will in turn trigger the conversion of the Google spreadsheet. The page is connected "live" to the spreadsheet.

> ⚠ **Important** : once your configuration is ready, do NOT leave Sparnatural pointing to the live spreadsheet, otherwise your page will depend on the availability of the online converter. Instead, save the result of the conversion to a local file "sparnatural-config.ttl", and adjust the "src" attribute of the <spar-natural> HTML element to point to the local file.

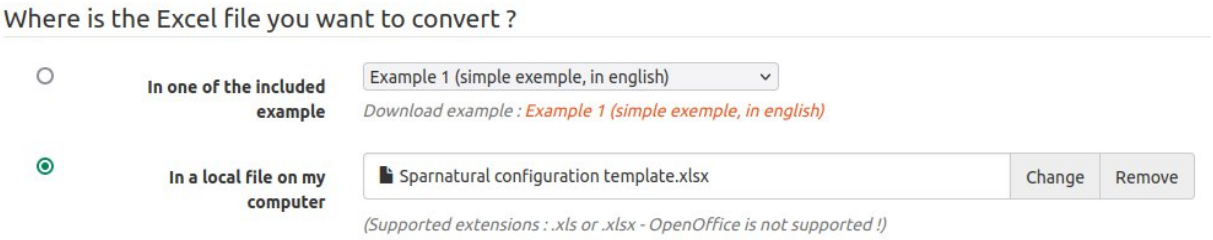
If you use a local spreadsheet

Relying on Google services might not be applicable in every context. It is also possible to design the configuration in a local spreadsheet, and convert it to an OWL file. The configuration is not live in that case, and you will have to reconvert the file every time you make a change in it.

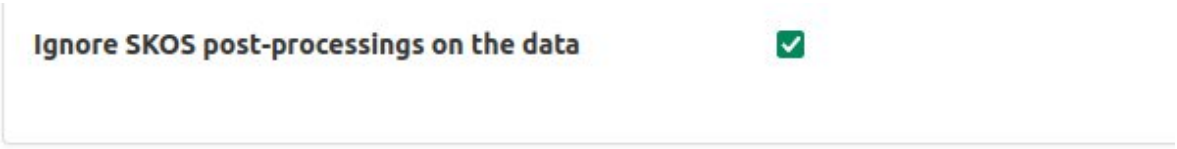
To start a fresh configuration template:

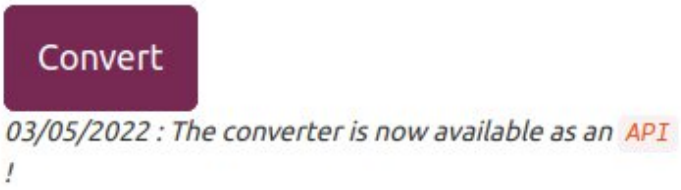
1. Download the configuration spreadsheet template.
2. Edit the content as necessary
3. Go to the online converter at https://skos-play.sparna.fr/play/convert

4. Upload the file in the field “in a local file on my computer”:



5. Check the box “Ignore SKOS post-processings on the data”:





6. Click on Convert.
7. Save the resulting file in the same folder as your Sparnatural page.
8. Adjust the “src” attribute of the <spar-natural> HTML element to point to this local file.

Reconvert the file the same way every time you make a change in it.

Filling-in the configuration spreadsheet

In this documentation we will work with a local spreadsheet. Download the spreadsheet configuration template and save it in a local file. You will be working on this local file.

Important : throughout this documentation, we are referring to the columns of the spreadsheet by their header name. The header is the green line in bold:

	the header	*owl:Class*	*corresponds to a literal value*	*class*	*needed.*	*set of icons.*
15	**URI**	**rdf:type**	**rdfs:subClassOf**	**rdfs:label@en**	**rdfs:label@fr**	**core:falcon**

Each column header corresponds to one configuration property as detailed in the Sparnatural OWL configuration reference documentation. The header line does not need to be at a fixed line; it is automagically detected, so don’t worry if you add or delete lines before this one.

Adjusting the ontology URI and the prefixes

You first need to adjust the URI of your ontology, as well as enter the prefixes used in your knowledge graph.

Ontology IRI

Make sure you are on the “classes” tab of the configuration template, and edit the content of cell B1. This cell needs to contain the URI of your configuration ontology. It is not very important, unless you plan to share your configuration later. It is typically set to something like “https://data.mydomain.com/sparnatural-config” or to a URL where Sparnatural will be deployed, like “https://mydomain.com/sparnatural-page/sparnatural-config”.

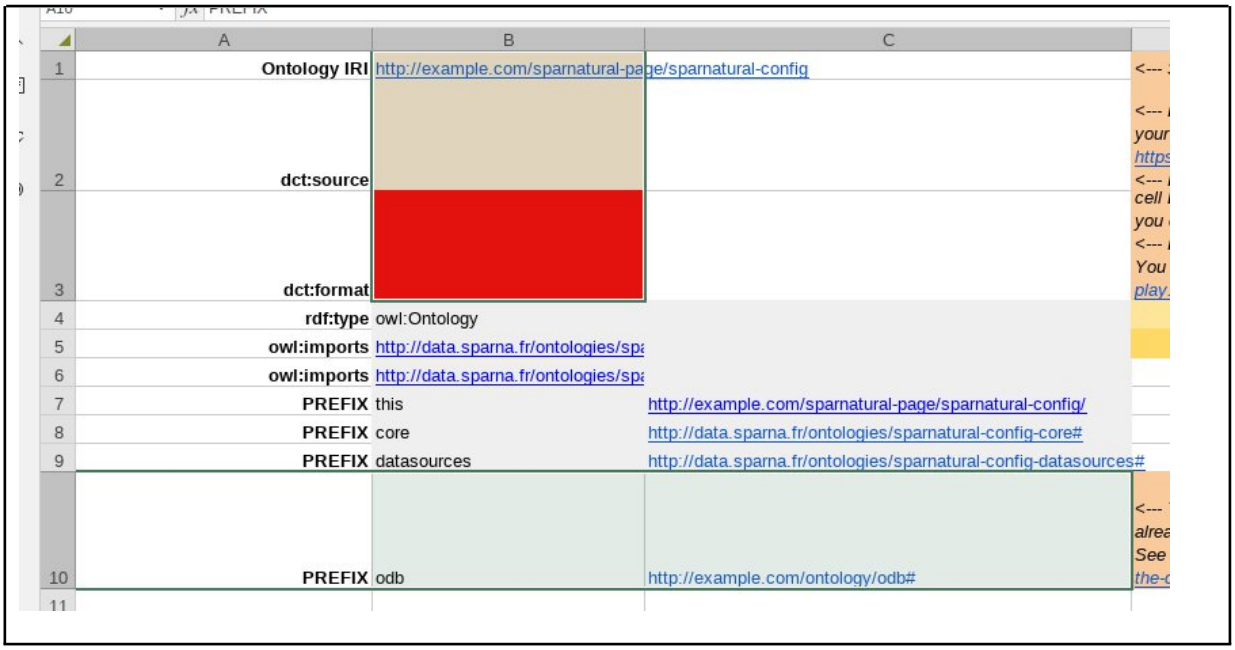
Metadata cleanup

Cells B2 and B3 are only useful when working with online Google spreadsheets, so that the configuration can be automated. We don’t need that in a local file, so simply delete the content of cells B2 and B3. Keep them if you work with a Google spreadsheet.

Prefixes

You need to add additional prefixes from your ontology. Some prefixes are already declared : “this”, “core” and “datasources”. Leave them as they are, and add prefixes in the same way in the lines below. The column A always needs to contain the keyword PREFIX, column B is the prefix name, and column C is the complete URI associated with the prefix.
Don’t hesitate to add new lines if you need to add many prefixes.

> *Example*
>
> Following the above, in our example configuration we set the Ontology IRI to http://example.com/sparnatural-page/sparnatural-config, delete the content of cells B2 and B3, and add our prefix “odb” on line 10, corresponding to the URI http://example.com/ontology/odb#



	A	B	C
1	Ontology IRI	http://example.com/sparnatural-page/sparnatural-config	
2	dct:source		
3	dct:format		
4	rdf:type	owl:Ontology	
5	owl:imports	http://data.sparna.fr/ontologies/spa	
6	owl:imports	http://data.sparna.fr/ontologies/spa	
7	PREFIX	this	http://example.com/sparnatural-page/sparnatural-config/
8	PREFIX	core	http://data.sparna.fr/ontologies/sparnatural-config-core#
9	PREFIX	datasources	http://data.sparna.fr/ontologies/sparnatural-config-datasources#
10	PREFIX	odb	http://example.com/ontology/odb#

Declaring classes

Now you can start filling in the table with the classes of your ontology. Don't hesitate to read the guidelines in the green line above the body of the table.

- use the prefix you declared first to write down the URIs you have in the URI column ;
- then in the rdf:type column set owl:Class as the value of all your classes items ;
- set all your classes as core:SparnaturalClass in the column rdfs:subClassOf.
- then add the label of your classes, in the rdfs:label@xx column (these will appear as the coloured named "blocks" in the query builder).

Advanced note: You can change the language of the label by editing the header row. By default the template enables labels in english (rdfs:label@en), and french (rdfs:label@fr). You can adjust the language code after the "@" sign. All the labels in a given column will be tagged with this language. Make sure the language you use matches the "lang" parameter of Sparnatural in your webpage. More on this in the section about multilingual configuration.

Next two columns allows to customize the display of the classes in the query builder :

- the core:falcon (as for "FontAwesome icon") column is where you can copy-paste the code of a Font Awesome free icon you will choose on the website (e.g. "fa-solid fa-car") ;
- if you need some, you can also add tooltips in the core:tooltip@en column. This is not mandatory. Depending on the use-case, the tooltip may provide more contextual

information to the user than only the definition from the ontology (e.g. *“Select this entry if you want to search on xxx or yyyy”*).

- Similar to labels, you can adjust the language code of the tooltips by editing the language code after the “@” symbol in the header line.

> **Tip**: HTML markup is supported in tooltips.

- In column core:order^^xsd:integer set the display order of each entry to sort the items in Sparnatural’s interface. The value must be an integer.

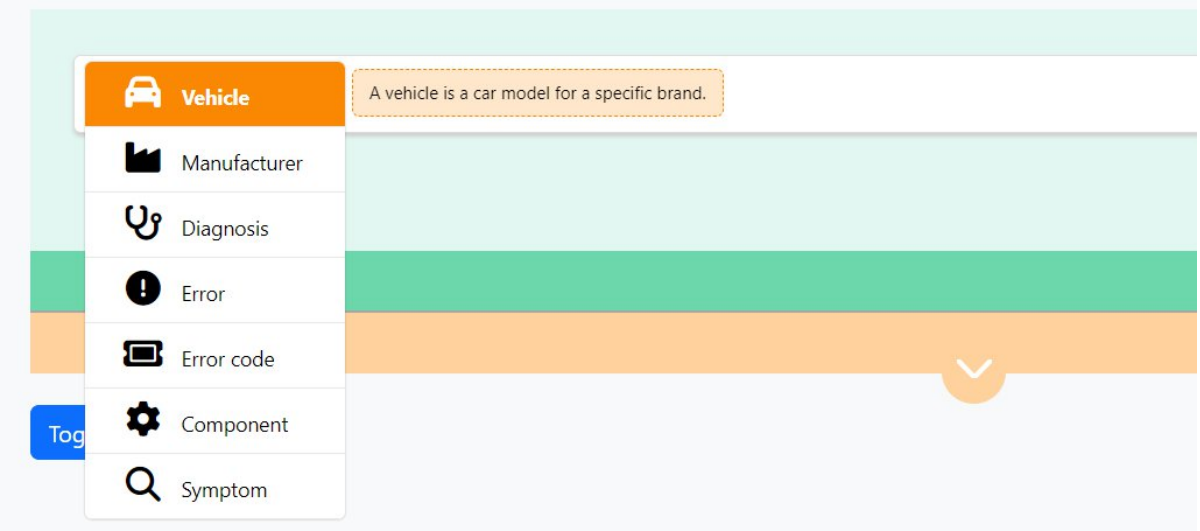
> **Tip**: By using the labels combined with the order, you can group your classes in a meaningful way, for example by setting a label that contains a hierarchy, such as “Actor > Person” and “Actor > Organization”, and setting those 2 classes next to each other with their order.

Example

Here in the example we have chosen to list all the existing classes of the model (you could choose to have only some classes of your model, and not all). We took the same URIs as the ones in the data model and added labels, icons, tooltips and order :

	URI of the class. This column can use prefixes declared above in the header	*This should **always** be owl:Class*	*This should be set to core:SparnaturalClass most of the time, or rdfs:Literal when it corresponds to a literal value*	*English label of the class*	*The Fontawesome icon code for the class, e.g. "fa-duotone fa-user". Search for icon codes at https://fontawesome.com/. Fontawesome provides a limited number of icons for free, and you can buy a license to access the full set of icons.*	*The english tooltip for the class.*	*The sort order of the class in the class dropdown list. This is an integer, e.g. "1", "2", etc.*
12							
13	**URI**	**rdf:type**	**rdfs:subClassOf**	**rdfs:label@en**	**core:falcon**	**core:tooltip@en**	**core:order^^ xsd:integer**
14	odb:Manufacturer	owl:Class	core:SparnaturalClass	Manufacturer	fa-solid fa-industry	A car manufacturer is a company whos	2
15	odb:Vehicle	owl:Class	core:SparnaturalClass	Vehicle	fa-solid fa-car	A vehicle is a car model for a specific l	1
16	odb:Diagnostic	owl:Class	core:SparnaturalClass	Diagnosis	fa-solid fa-stethoscope	A diagnosis identifies a possible proble	3
17	odb:Error	owl:Class	core:SparnaturalClass	Error	fa-solid fa-circle-exclamation	An error is an element that comes up c	4
18	odb:ErrorCode	owl:Class	core:SparnaturalClass	Error code	fa-solid fa-ticket	An error code is a set of numbers follo	5
19	odb:Symptom	owl:Class	core:SparnaturalClass	Symptom	fa-solid fa-magnifying-glass	A symptom is a phenomenon, percepti	7
20	odb:Component	owl:Class	core:SparnaturalClass	Component	fa-solid fa-gear	A class representing a component of a	6
21							
22	this:Attribute	owl:Class	rdfs:Literal	Attribute	fa-solid fa-pen-to-square	A class to display literal values (as example : text, boo	
23							
24							

We decided that “Vehicle” was an important entry point and set its order to 1. Following this, we can see it appears first in the query builder :



Note how the tooltip displays the definition from the ontology.

Declaring properties

Same process then to set the relations between the classes : jump to the “Properties” tab, 2nd of the spreadsheet.

Tip: We suggest you organize this table by sections, each section corresponding to the specification of the properties attached to one given class in your configuration. Make a colored line for each section, with the name of the class as the title. Generally you are free to arrange the spreadsheet as you want and use any formatting/color option you want. Lines that do no contain a URI in column A will be ignored.

In this tab you will enter:

- URI column : URI of your property, typically using a prefix from your ontology ;
- in the rdf:type column always set the value to owl:ObjectProperty ;

  Advanced note: even when configuring properties that actually correspond to datatype properties, you always have to use owl:ObjectProperty, as for Sparnatural the property needs to have a domain and a range that are classes.

- in the “rdfs:label@en” column set the label of the property to be shown in the interface ;
  - adjust the language code of the labels by editing the language code after the “@” symbol in the header line.
- the rdfs:subPropertyOf column is used to configure the way the values can be selected in the query builder (see “widget” section below) : when you start designing

your configuration we suggest using `core:ListProperty` to obtain simple populated lists using the data ; you can then refine this to other more appropriate values after.

- if needed a tooltip in the core:tooltip@en column ;
    - adjust the language code of the tooltips by editing the language code after the "@" symbol in the header line.

And in order to relate each property to its domain class and its range class:

- the rdfs:domain is the Class to which the property is assigned (as the "subject" of the assertion in an RDF graph) ;
- the rdfs:range is the Sparnatural Class to which the property points to (the "object" of an RDF predicate);

These 2 columns must refer to a URI of a class from the first tab of your configuration spreadsheet.

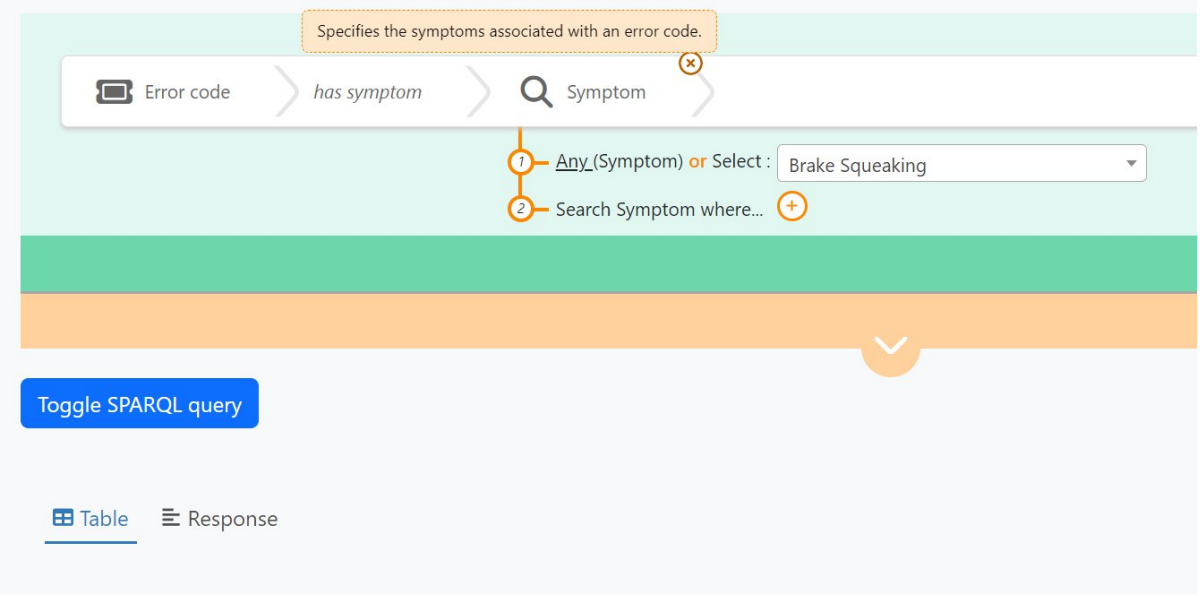
 **Advanced note**: it is possible that a single property has more than one class as its domain or its range. You can specify more than one class identifier in the rdfs:domain or rdfs:range column, by separating them with a comma.

Example

Note how the table is organized with one section per class; note also how each property refers to the class to which it is attached in the rdfs:domain column (in each "section" the rdfs:domain is always the same), and the class to which it refers to in the rdfs:range column.

	A	B	C	E	F	H	I
1	**Ontology IRI**	https://data.mydomai	<--- Don't touch this cell				
2							
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*This must **always** be owl:ObjectProperty*	*English label of the property*	*Indicates the widget type of the property. This can take its value in one of the predefined sparnatural property types.*	*The english tooltip for the property.*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes*
4	**URI**	**rdf:type**	**rdfs:label@en**	**rdfs:subPropertyOf**	**core:tooltip@en**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**
5	Manufacturer						
6	odb:name	owl:ObjectProperty	has name	core:NonSelectableProperty	Specifies the name of the manufacturer.	odb:Manufacturer	this:Attribute
7	Vehicle						
8	odb:VIN	owl:ObjectProperty	has VIN	core:AutocompleteProperty	Specifies the Vehicle Identification Number (VIN) of the vehicle.	odb:Vehicle	this:Attribute
9	odb:hasManufacturer	owl:ObjectProperty	has manufacturer	core:ListProperty	Specifies the manufacturer of the vehicle.	odb:Vehicle	odb:Manufacturer
10	this:hasDiagnosis	owl:ObjectProperty	has diagnosis	core:NonSelectableProperty	The property is the inverse of odb:analysedVehicle.	odb:Vehicle	odb:Diagnostic
11	Diagnostic						
12	odb:diagnosticDate	owl:ObjectProperty	has diagnosis date	core:TimeProperty-Date	Defines the date on which the diagnosis occurs.	odb:Diagnostic	this:Attribute
13	odb:analysedVehicle	owl:ObjectProperty	analysed vehicle	core:AutocompleteProperty	Specifies that the vehicle has been analyzed, to identify a potential problem.	odb:Diagnostic	odb:Vehicle
14	odb:hasResults	owl:ObjectProperty	has results	core:NonSelectableProperty	Specifies the results, from the analysis.	odb:Diagnostic	odb:Error
15	this:returnsCode	owl:ObjectProperty	returns code	core:ListProperty	The property is a shortcut between Diagnosis and Error Code.	odb:Diagnostic	odb:ErrorCode
16	Error						
17	odb:alreadyRaised	owl:ObjectProperty	already raised	core:BooleanProperty	Attribute indicating whether an error has already been detected previously.	odb:Error	this:Attribute
18	odb:hasErrorCode	owl:ObjectProperty	has error code	core:ListProperty	Specifies the error code relating to an error reported during a diagnostic.	odb:Error	odb:ErrorCode
19	ErrorCode						
20	odb:hasSymptom	owl:ObjectProperty	has symptom	core:ListProperty	Specifies the symptoms associated with an error code.	odb:ErrorCode	odb:Symptom
21	odb:hasComponent	owl:ObjectProperty	has component	core:ListProperty	Specifies the components impacted by an error code.	odb:ErrorCode	odb:Component
22	Symptom						
23	this:symptomLabel	owl:ObjectProperty	label	core:SearchProperty	Specifies the name of the object.	odb:Symptom	this:Attribute
24	Component						
25	odb:componentCode	owl:ObjectProperty	has component co	core:SearchProperty	Specifies the unique code of the component.	odb:Component	this:Attribute
26	this:componentLabel	owl:ObjectProperty	label	core:SearchProperty	Specifies the name of the object.	odb:Component	this:Attribute
	this:labelOrCode	owl:ObjectProperty	label or code	core:SearchProperty	Allows to get a label or a code.	odb:Component	this:Attribute

As a result we can see - when index.html is refreshed - the object properties *in italic* appear in the interface, between the classes items :



The tooltip of the property is displayed if it was added before in the configuration file.

We see a dropdown list appears when the range of the query (i.e. the “object” class of the assertion) is chosen. As explained before, the way the selected values are to be displayed depends on the type (rdfs:subPropertyOf) of the property, also referred to as the “widget” of the property.

Selecting property types (widgets)

For now Sparnatural offers the following ways of selecting a value for a criteria :

Widget type (rdfs:subPropertyOf)	Description
core:ListProperty (or core:LiteralListProperty which is deprecated)	dropdown list widget
core:AutocompleteProperty	autocomplete search field
core:TreeProperty	tree browsing widget, useful with some tree-shaped values, typically SKOS hierarchies, part-of hierarchies, etc;
core:MapProperty	map selection widget (GeoSPARQL queries)
core:SearchProperty, core:StringEqualsProperty, core:GraphDBSearchProperty	string search widget, searched as regex or as exact string
core:TimeProperty-Date, core:TimeProperty-Year	date range widget (date or year precision)
core:BooleanProperty	boolean widget (true/false, yes/no values…)
core:NonSelectableProperty	no value selection (useful for 'intermediate' entities whose values don't need to be displayed)

All of them are already fully documented in the reference documentation for Sparnatural widgets .

The choice of the widget is driven by how we want the user to select a value, and how many different values are available (e.g. lists are good only when the values are relatively small, typically less than 500 distinct values).

Example

Note how the properties in our configuration uses different kinds of widgets:

	URI	rdf:type	rdfs:label@en	rdfs:label@fr	rdfs:subPropertyOf
4					
5	Manufacturer				
6	odb:name	owl:ObjectProperty	has name	nom	core:NonSelectableProperty
7	Vehicle				
8	odb:VIN	owl:ObjectProperty	has VIN	a pour VIN	core:AutocompleteProperty
9	odb:hasManufacturer	owl:ObjectProperty	has manufacturer	a pour constructeur	core:ListProperty
10	this:hasDiagnosis	owl:ObjectProperty	has diagnosis	a pour diagnostic	core:NonSelectableProperty
11	Diagnostic				
12	odb:diagnosticDate	owl:ObjectProperty	has diagnosis date	date du diagnostic	core:TimeProperty-Date
13	odb:analysedVehicle	owl:ObjectProperty	analysed vehicle	véhicule analysé	core:AutocompleteProperty
14	odb:hasResults	owl:ObjectProperty	has results	a pour résultat	core:NonSelectableProperty
15	this:returnsCode	owl:ObjectProperty	returns code	renvoie le code	core:ListProperty
16	Error				
17	odb:alreadyRaised	owl:ObjectProperty	already raised	déjà signalée	core:BooleanProperty
18	odb:hasErrorCode	owl:ObjectProperty	has error code	a pour code d'erreur	core:ListProperty
19	ErrorCode				
20	odb:hasSymptom	owl:ObjectProperty	has symptom	a pour symptôme	core:ListProperty
21	odb:hasComponent	owl:ObjectProperty	has component	concerne le composant	core:ListProperty
22	Symptom				
23	this:symptomLabel	owl:ObjectProperty	label	a pour libellé	core:SearchProperty
24	Component				
25	odb:componentCode	owl:ObjectProperty	has component code	a pour code composant	core:SearchProperty
26	this:componentLabel	owl:ObjectProperty	label	a pour libellé	core:SearchProperty
27	this:labelOrCode	owl:ObjectProperty	label or code	a pour libellé ou code	core:SearchProperty

On Manufacturer, we have set the `odb:name` property as `core:NonSelectableProperty`, because we assume the user will never have to search or select a value for the name of a Manufacturer.
On Vehicle, the `odb:VIN` property is set as an autocomplete. Being a long technical identifier, having an autocomplete will help user selecting a correct value. The `odb:manufacturer` property uses a `core:ListProperty` because there is a limited list of possible car manufacturers, so using a list is convenient.
On Diagnostic, `odb:diagnosticDate` uses a date property as the values in the graph have an `xsd:date` datatype.

Populating lists and autocomplete fields (datasources)

Using predefined datasources

ListProperty and AutocompleteProperty require a datasource to be populated correctly. For that purpose use the datasources:datasource column of the Properties tab. The datasource of a dropdown list populates the list, the datasource of an autocomplete property feeds the autocomplete proposals. TreeProperty also requires two datasources; the configuration of tree datasources is covered in annex.

In its most simple form, a datasource is a SPARQL query that will return some results.

Sparnatural comes with off-the-shelves datasources, in tab “sparnatural-config-core” of the spreadsheet. Here you can find a list of preconfigured datasources corresponding to different widget types for lists, autocomplete (search) and tree.

List of possible widget types	List of preconfigured datasources	List of preconfigured queries
core:AutocompleteProperty	datasources:list_dctermstitle_alpha	datasources:query_list_label_alpha
core:ListProperty	datasources:list_dctermstitle_count	datasources:query_list_label_count
core:TimeProperty-Date	datasources:list_dctermstitle_alpha_with_count	datasources:query_list_label_alpha_with_count
core:TimeProperty-Year	datasources:list_foafname_alpha	datasources:query_list_label_with_range_alpha
core:SearchProperty	datasources:list_foafname_count	datasources:query_list_label_with_range_alpha_with_count
core:GraphDBSearchProperty	datasources:list_foafname_alpha_with_count	datasources:query_list_label_with_range_count
core:NonSelectableProperty	datasources:list_rdfslabel_alpha	datasources:query_list_URI_alpha
core:LiteralListProperty	datasources:list_rdfslabel_count	datasources:query_list_URI_count
core:BooleanProperty	datasources:list_rdfslabel_alpha_with_count	datasources:query_list_URI_or_literal_alpha
core:StringEqualsProperty	datasources:list_schemaname_alpha	datasources:query_list_URI_or_literal_alpha_with_count
core:TreeProperty	datasources:list_schemaname_count	datasources:query_list_URI_or_literal_count
	datasources:list_schemaname_alpha_with_count	datasources:query_literal_list_alpha
	datasources:list_skospreflabel_alpha	datasources:query_literal_list_alpha_with_count
	datasources:list_skospreflabel_count	datasources:query_literal_list_count
	datasources:list_skospreflabel_alpha_with_count	datasources:query_search_label_bifcontains
	datasources:list_URI_alpha	datasources:query_search_label_contains
	datasources:list_URI_count	datasources:query_search_label_strstarts
	datasources:list_URI_or_literal_alpha	datasources:query_search_literal_contains
	datasources:list_URI_or_literal_alpha_with_count	datasources:query_search_literal_strstarts
	datasources:list_URI_or_literal_count	datasources:query_search_URI_contains
	datasources:literal_list_alpha	datasources:query_tree_children
	datasources:literal_list_alpha_with_count	datasources:query_tree_children_with_count
	datasources:literal_list_count	datasources:query_tree_root_noparent
	datasources:search_dctermstitle_bifcontains	datasources:query_tree_root_noparent_with_count
	datasources:search_dctermstitle_contains	datasources:query_tree_root_domain
	datasources:search_dctermstitle_strstarts	
	datasources:search_foafname_bifcontains	
	datasources:search_foafname_contains	
	datasources:search_foafname_strstarts	
	datasources:search_rdfslabel_bifcontains	
	datasources:search_rdfslabel_contains	
	datasources:search_rdfslabel_strstarts	
	datasources:search_schemaname_bifcontains	
	datasources:search_schemaname_contains	

The predefined datasources are documented in the datasource documentation of Sparnatural, but we give some simple indications to select the adequate one for your use-case:

- datasources beginning by “list” are for ListProperty, while datasources beginning by “search” are for AutocompleteProperty.
- The identifier of the property indicates which property Sparnatural uses to display the entry or search on it : rdfs:label, foaf:name, dcterms:title, schema:name, skos:prefLabel
- List datasources come in 3 variants : “alpha” is pure alphabetical, count is sorted by descending number of occurrences, “alpha_with_count” is alphabetical but displays the number of occurrences in parenthesis.
- Search datasources come in 3 variants : “strstarts” looks for the string at the beginning of the property, “contains” looks for the string anywhere in the property, “bifcontains” is specific to Virtuoso and will look for the string anywhere in the property but as a complete word/token.

A typical frequent choice to populate a list is the datasource “datasource:list_rdfslabel_alpha” which will populate a list with the rdfs:label of the values, sorted alphabetically.

Advanced note: if you look at the SPARQL queries (e.g. by navigating to the URI of one "query_list_xxxx"), you will notice that the default provided queries do not use the range class as a criteria in the query, mostly for performance reasons. They assume that a given property always refers to a single type of entity. If you have a property that can refer to multiple classes as range, then you need to use one of the provided query that includes "with_range" in its name (e.g. `datasources:query_list_label_with_range_alpha`), and inject the property name in it (see following section)

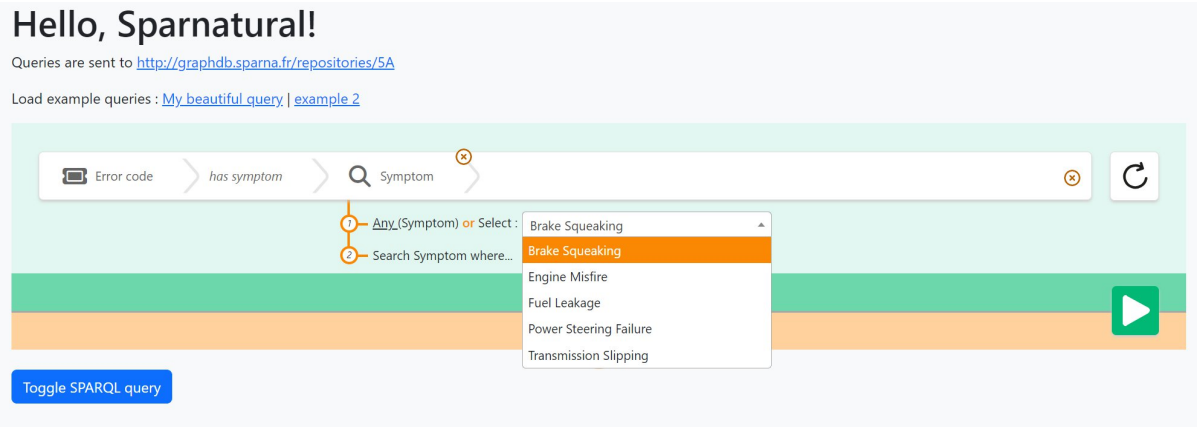
Advanced note: if you don't specify any datasource, Sparnatural will default to datasources:list_URI_or_literal_alpha for lists and to datasources:search_URI_contains or datasources:search_literal_contains (depending if the range class is marked as a literal or not, see below). You will most probably never use these defaults and always specify a datasource.

Example

Both lines in grey below correspond to list properties ("`core:ListProperty`") "`hasSymptom`" and "`hasComponent`" respectively with "Symptom"and "Component" as range values, where we wanted the `rdfs:label` in an alphabetical way to be displayed :

	A	C	E	G	H	I
1	**Ontology IRI**	<--- Don't touch this cell				
2						
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*English label of the property*	*Indicates the widget type of the property. This can take its value in one of the predefined sparnatural property types.*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes can be given, separated by commas.*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes can be given, separated by commas.*	*A reference to a datasource, either a custom one from the "Datasources" tab or a provided one in the "sparnatural-config-core" tab. The datasource indicates how to populate the dropdown list*
4	**URI**	**rdfs:label@en**	**rdfs:subPropertyOf**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**	**datasources:datasource**
17	ErrorCode					
18	odb:hasSymptom	has symptom	core:ListProperty	odb:ErrorCode	odb:Symptom	datasources:list_rdfslabel_alpha
19	odb:hasComponent	has component	core:ListProperty	odb:ErrorCode	odb:Component	datasources:list_rdfslabel_alpha
20	Symptom					
21	rdfs:label	label	core:SearchProperty	odb:Symptom	this:Attribute	
22	Component					
23	odb:componentCode	has component code	core:SearchProperty	odb:Component	this:Attribute	
24	rdfs:label	label	core:SearchProperty	odb:Component	this:Attribute	

That gives us the following result when shown in Sparnatural's interface :



We can see that we obtain an alphabetically-sorted list of labels here (instead of URIs).

Using predefined queries with your own properties

When your data model uses a property to label entities other than one of the 5 for which preconfigured datasources exist, you can create your custom one, based on one of the predefined query (alpha, count or alpha_with_count), in which your property will be “injected”.

To do so, go to “Datasources” tab of your spreadsheet and write down the URI of the new datasource you want to create in column A, using the “this:” namespace, using a name as explicit as possible. Then:

- in rdf:type column, always set the value `datasources:SparqlDatasource`
- In the datasources:queryTemplate column, pick one of the query from the sparnatural-config-core tab you will copy-paste in the corresponding column. The queries identifiers start with “`datasources:query_list...`” or “`datasources:query_search...`”
- In the datasources:labelProperty column, enter the URI of the label property in your data, either as a complete URI (surrounded by “<” “>”) or as a prefixed one. Your custom datasource is created, and can refer to its URI from the “Properties” tab in the “datasources:datasource” column.

Example

Two examples of custom datasources here in the screenshot : first one to populate a simple list property with the `odb:name` label (alphabetical order), second one to trigger a “strstarts” search on VIN number labels for an autocomplete field property :

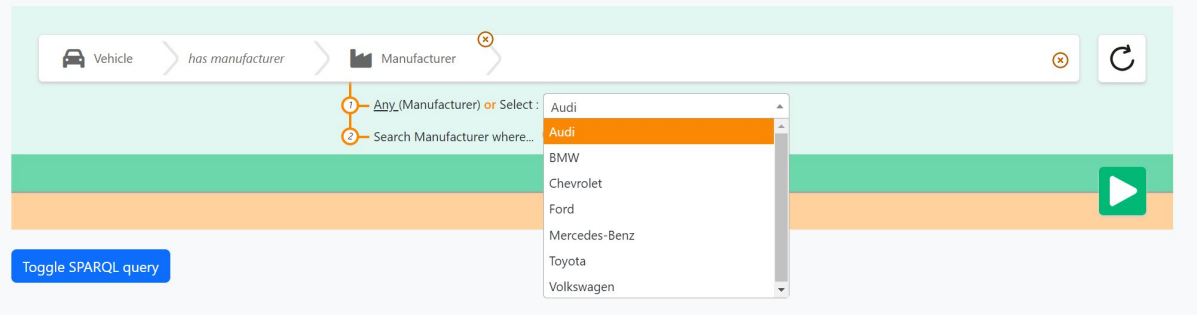
	A	C	E	G	H	I
1	**Ontology IRI**	<--- Don't touch this cell				
2						
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*English label of the property*	*Indicates the widget type of the property. This can take its value in one of the predefined sparnatural property types.*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes can be given, separated by commas.*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes can be given, separated by commas.*	*A reference to a datasource, either a custom one from the "Datasources" tab or a provided one in the "sparnatural-config-core" tab. The datasource indicates how to populate the dropdown list*
4	**URI**	**rdfs:label@en**	**rdfs:subPropertyOf**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**	**datasources:datasource**
5	Manufacturer					
6	odb:name	has name	core:NonSelectableProp	odb:Manufacturer	this:Attribute	
7	Vehicle					
8	odb:VIN	has VIN	core:AutocompleteProp	odb:Vehicle	this:Attribute	
9	odb:hasManufacturer	has manufacturer	core:ListProperty	odb:Vehicle	odb:Manufacturer	this:list_odbname_alpha
10	Diagnostic					
11	odb:diagnosticDate	has diagnosis date	core:TimeProperty-Date	odb:Diagnostic	this:Attribute	
12	odb:analysedVehicle	analysed vehicle	core:AutocompleteProp	odb:Diagnostic	odb:Vehicle	this:search_VIN_strstarts
13	odb:hasResults	has results	core:NonSelectableProp	odb:Diagnostic	odb:Error	

And this is what both custom datasources look like in the Datasources tab:

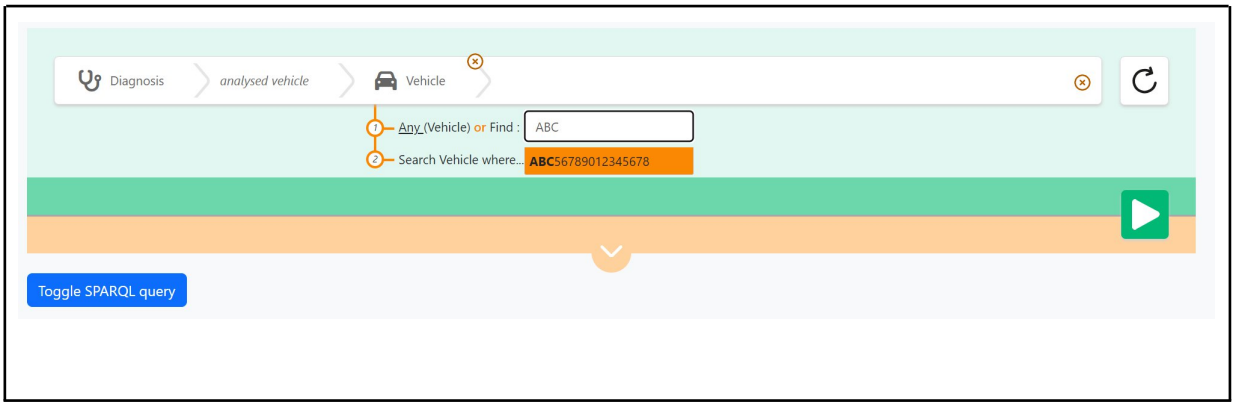
1	**Ontology IRI**	https://data.mydomain.com/ontol	<--- Don't touch this cell		
2					
3	*URI of the datasource in the configuration. This is the value that should be referenced from the "datasources:datasource" column in the properties tab*	*This must **always** be datasources:SparqlDatasource*	*Contains the query string, containing specific Sparnatural variables. See http://docs.sparnatural.eu/OWL-based-configuration-datasources.html#your-own-sparql-query-lists--autocomplete*	*A reference to the query template that this datasource relies on. use only when the query is reused for the same label property. Use EITHER the queryString column, OR the queryTemplate column, but not both.*	*Only if you used datasources:queryTemplate, the label property to inject into the query template. This must be a valid complete URI, including "<...>".*
4	**URI**	**rdf:type**	**datasources:queryString**	**datasources:queryTemplate**	**datasources:labelProperty**
5	this:list_myname_count	datasources:SparqlDatasource		datasources:query_list_label_count	<http://example.com/ontology/myname>
6	this:list_skosprefLabel_alpha_with_count_langfr	datasources:SparqlDatasource		this:query_list_label_alpha_with_count_langfr	skos:prefLabel
7	this:list_odbname_alpha	datasources:SparqlDatasource		datasources:query_list_label_alpha	odb:name
8	this:search_VIN_strstarts	datasources:SparqlDatasource		datasources:query_search_label_strstarts	odb:VIN

This way we obtain the following results on Sparnatural index page :

a dropdown list with particular labels instead of URIs :



an autocomplete field that could be filled in with the VIN numbers instead of the labels :



Declaring literal classes

You will have cases when a property is not an “object property” (i.e. a property followed by another resource as a value), but a “data property”. There you may have to deal with literal data as values, typically xsd:string, xsd:boolean, xsd:date or xsd:dateTime.

Sparnatural configuration allows you to create special classes dedicated to literal data in order to enable the display of these particular values the same as other classes.

For that purpose you need to create a range class corresponding to the literal values you want to display. The only two differences with other classes is that:

1. the rdfs:subClassOf column must have the value `rdfs:Literal` instead of the usual `core:SparnaturalClass` (and don’t forget to add other attributes to the class : label, icon, tooltip if needed etc.)
2. you will use the “this:” namespace as the URI of this class

Tip: Either you can declare a single class for all literal values, such as “this:Attribute”, so that all literal properties are “grouped” under a generic “Attribute” entry, or you can choose to decompose by datatype, such as “Text”, “Date”, “Boolean”, or you can even decompose by properties, with one literal class per literal property (e.g. “Coverage” class corresponding to “coverage” property), which imply some kind of duplication. The strategy to use depends on how you would like things to be presented to your users.

The consequence of declaring a class as `rdfs:Literal` is that the generated SPARQL query will never contain an `rdf:type` criteria for such objects, since they are literal values.

Remember that literal classes won’t appear in the initial classes menu as they will never be used as the domain of other properties (only as range).

Example

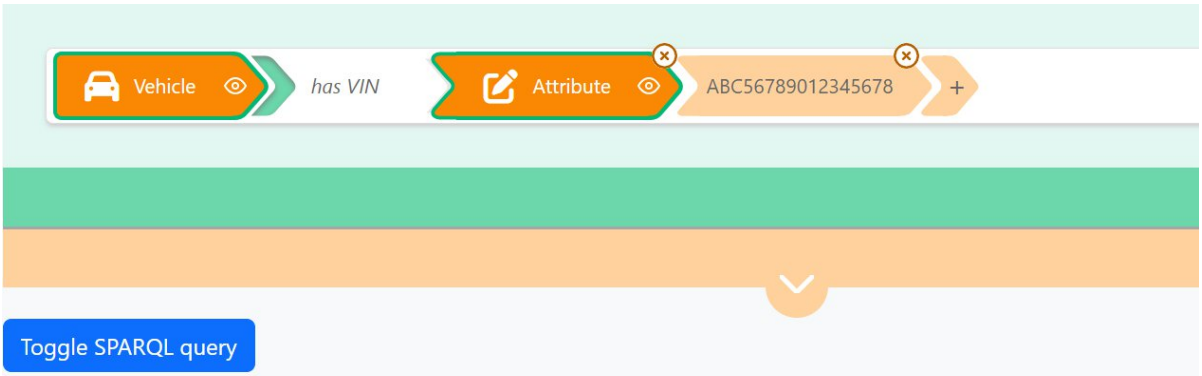
Back to the Classes tab, a view of the this:Attribute class (blue line) that will be used as a range class each time a property is to display literal values : as the class doesn't really exist in the data, it is provided a "this" URI, and has the value rdfs:Literal in the rdfs:subClassOf column :

	URI	rdf:type	rdfs:subClassOf	rdfs:label@en
13				
14	odb:Manufacturer	owl:Class	core:SparnaturalClass	Manufacturer
15	odb:Vehicle	owl:Class	core:SparnaturalClass	Vehicle
16	odb:Diagnostic	owl:Class	core:SparnaturalClass	Diagnosis
17	odb:Error	owl:Class	core:SparnaturalClass	Error
18	odb:ErrorCode	owl:Class	core:SparnaturalClass	Error code
19	odb:Symptom	owl:Class	core:SparnaturalClass	Symptom
20	odb:Component	owl:Class	core:SparnaturalClass	Component
21				
22	this:Attribute	owl:Class	rdfs:Literal	Attribute
23				
24				

This way the corresponding literal properties are all pointing to the this:Attribute class as a range cf. rdfs:range column :

	A	C	E	G	H
1	**Ontology IRI**	<--- Don't touch this cell			
2					
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*English label of the property*	*Indicates the widget type of the property. This can take its value in one of the predefined sparnatural property types.*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes can be given, separated by commas.*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes can be given, separated by commas.*
4	**URI**	**rdfs:label@en**	**rdfs:subPropertyOf**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**
5	Manufacturer				
6	odb:name	has name	core:NonSelectableProperty	odb:Manufacturer	this:Attribute
7	Vehicle				
8	odb:VIN	has VIN	core:AutocompleteProperty	odb:Vehicle	this:Attribute
9	odb:hasManufacturer	has manufacturer	core:ListProperty	odb:Vehicle	odb:Manufacturer
10	Diagnostic				
11	odb:diagnosticDate	has diagnosis date	core:TimeProperty-Date	odb:Diagnostic	this:Attribute
12	odb:analysedVehicle	analysed vehicle	core:AutocompleteProperty	odb:Diagnostic	odb:Vehicle
13	odb:hasResults	has results	core:NonSelectableProperty	odb:Diagnostic	odb:Error
14	Error				
15	odb:alreadyRaised	already raised	core:BooleanProperty	odb:Error	this:Attribute
16	odb:hasErrorCode	has error code	core:ListProperty	odb:Error	odb:ErrorCode
17	ErrorCode				
18	odb:hasSymptom	has symptom	core:ListProperty	odb:ErrorCode	odb:Symptom
19	odb:hasComponent	has component	core:ListProperty	odb:ErrorCode	odb:Component
20	Symptom				
21	rdfs:label	label	core:SearchProperty	odb:Symptom	this:Attribute
22	Component				
23	odb:componentCode	has component code	core:SearchProperty	odb:Component	this:Attribute
24	rdfs:label	label	core:SearchProperty	odb:Component	this:Attribute

Then searching in the query builder for the VIN number of a Vehicle, that is a literal attribute, you can see the query when clicking on blue “Toggle SPARQL query” button :




```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?Vehicle_1 ?Attribute_2 WHERE {
  ?Vehicle_1 rdf:type <http://example.com/ontology/odb#Vehicle>;
    <http://example.com/ontology/odb#VIN> ?Attribute_2.
  VALUES ?Attribute_2 {
    "ABC56789012345678"
  }
}
LIMIT 1000
```

Note how the query does *not* include an `rdf:type` criteria on `this:Attribute`.

How-to set some properties optional or negative

According to the SPARQL syntax, Sparnatural offers also a way to configure optional or negative assertions, corresponding in SPARQL to OPTIONAL or negative "FILTER NOT EXISTS" query patterns.

Both parameters can be activated/inactivated for each individual property in the Properties tab of the spreadsheet. If you set "true" as the value of the column `core:enableOptional^^xsd:boolean` or `core:enableNegation^^xsd:boolean`, a clickable green arrow will appear in the query builder interface before the chosen property, enabling the user to make the property criteria optional or negative.

Example

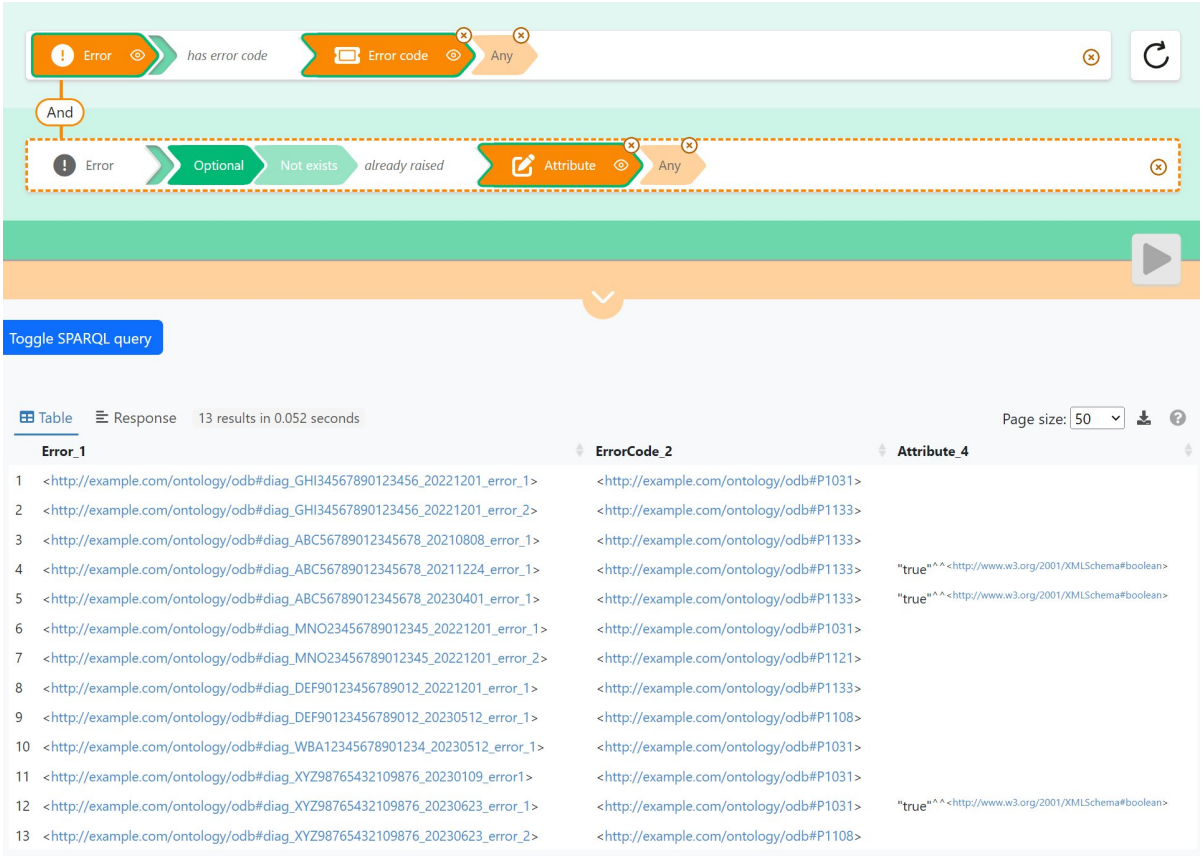
Here we can see in both last columns we have chosen to enable the Optional parameter for only one row (the "already raised" property) and a few more ones to set the negative parameter : theoretically you can apply both parameters to them all, but here we preferred allowing the option for relevant ones only (the choice depending on the existing data).

So regarding the optional parameter, the "already raised property" is the only one being facultative, so you may want to display optionally the existing values of it without excluding the blank ones in your query.

The negatives ones which are set on "TRUE" ("VRAI") are those for which a negative query was judged meaningful from a user perspective.

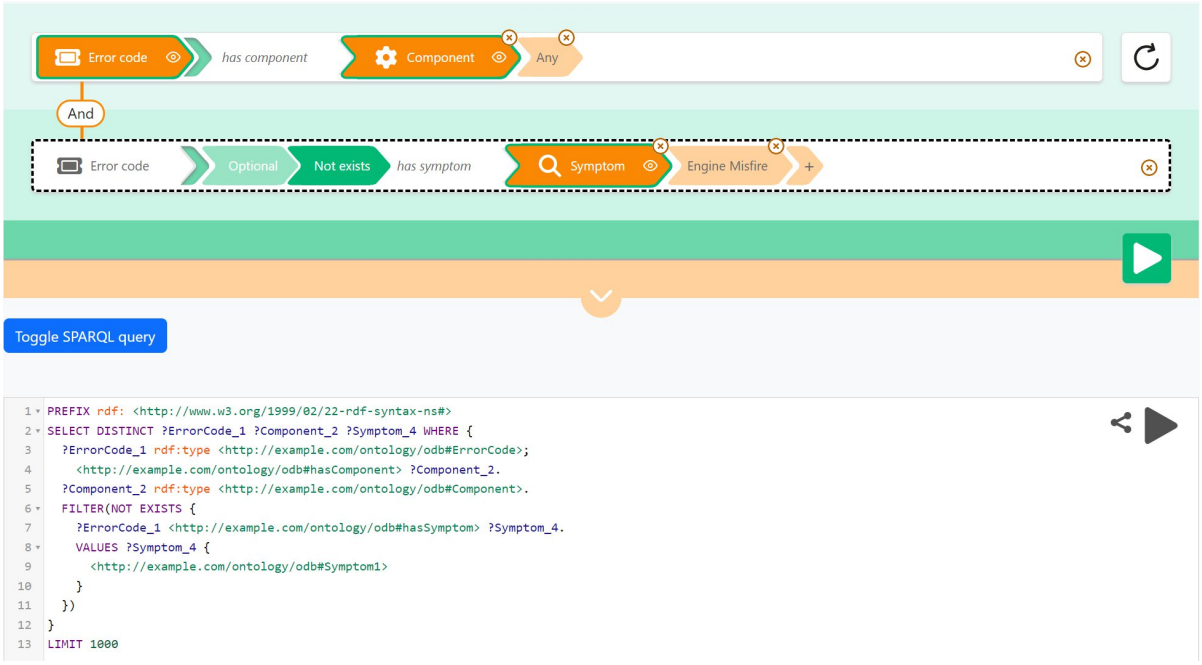
	URI	rdfs:label@en	rdfs:subPropertyOf	rdfs:domain(separator=",")	rdfs:range(separator=",")	core:enableOptional^^xsd:boolean	core:enableNegation^^xsd:boolean
4							
5	Manufacturer						
6	odb:name	has name	core:NonSelectableProperty	odb:Manufacturer	this:Attribute		
7	Vehicle						
8	odb:VIN	has VIN	core:AutocompleteProperty	odb:Vehicle	this:Attribute		
9	odb:hasManufacturer	has manufacturer	core:ListProperty	odb:Vehicle	odb:Manufacturer		VRAI
10	this:hasDiagnosis	has diagnosis	core:NonSelectableProperty	odb:Vehicle	odb:Diagnostic		VRAI
11	Diagnostic						
12	odb:diagnosticDate	has diagnosis date	core:TimeProperty-Date	odb:Diagnostic	this:Attribute		VRAI
13	odb:analysedVehicle	analysed vehicle	core:AutocompleteProperty	odb:Diagnostic	odb:Vehicle		
14	odb:hasResults	has results	core:NonSelectableProperty	odb:Diagnostic	odb:Error		
15	this:returnsCode	returns code	core:ListProperty	odb:Diagnostic	odb:ErrorCode		
16	Error						
17	odb:alreadyRaised	already raised	core:BooleanProperty	odb:Error	this:Attribute	VRAI	VRAI
18	odb:hasErrorCode	has error code	core:ListProperty	odb:Error	odb:ErrorCode		VRAI
19	ErrorCode						
20	odb:hasSymptom	has symptom	core:ListProperty	odb:ErrorCode	odb:Symptom		VRAI
21	odb:hasComponent	has component	core:ListProperty	odb:ErrorCode	odb:Component		VRAI
22	Symptom						
23	this:symptomLabel	label	core:SearchProperty	odb:Symptom	this:Attribute		
24	Component						
25	odb:componentCode	has component code	core:SearchProperty	odb:Component	this:Attribute		
26	this:componentLabel	label	core:SearchProperty	odb:Component	this:Attribute		
27	this:labelOrCode	label or code	core:SearchProperty	odb:Component	this:Attribute		

The following screenshot shows an optional query pattern on the “already raised” property which is optional (cardinality [0..1]). Let’s imagine we’d like to display all the results following this property no matter *if actually there are some* (or not). This enables to obtain a list of results even in case when the value isn’t there :



This one shows a negative pattern where we want to search for every component related to

an error code that does not have “Engine Misfire” as a symptom :



How-to map classes and properties to the underlying data model

By default, you use the URI identifiers of the classes and properties of your data model as the URI of classes and properties in your Sparnatural configuration. But you can also provide your users with a slightly different view of the underlying graph structure. Typically you might want to show them a simplified view of the more elaborate structure in the graph. To do this you will use different URI identifiers for classes and properties in your Sparnatural configuration, that will be remapped at query time to the underlying graph structure.

General mechanism

Declare the new URI identifiers using the “this:” namespace. This means that these identifiers belong only to your configuration, not to your knowledge graph ontology.

The mapping is done through the core:sparqlString column in the “Classes” and “Properties” tab. *The string that you specify in the core:sparqlString annotation will be inserted “as is” in the generated SPARQL query, in place of the corresponding property or class identifier.*

Warning : You need to be careful that the string you provide is a valid “piece of SPARQL”, otherwise the query will be syntactically wrong. The mappings for properties shall use the SPARQL property path syntax, please refer to this specification for all details. basically the core:sparqlString value for a property can be *any valid SPARQL property path*.

Warning : values of the core:sparqlString annotation must not use prefixed URIs, only full URIs, surrounded by “<...>”.

As an example, if your configuration uses a property URI “this:foo” that has a core:sparqlString value “<http://bar>”, then this is the string “<http://bar>” that will be in the final query, in place of “this:foo”.

Follow the “recipes” below that will guide you on how to write the content of the core:sparqlString column depending on your use-case.

Querying a sequence of properties (using a shortcut)

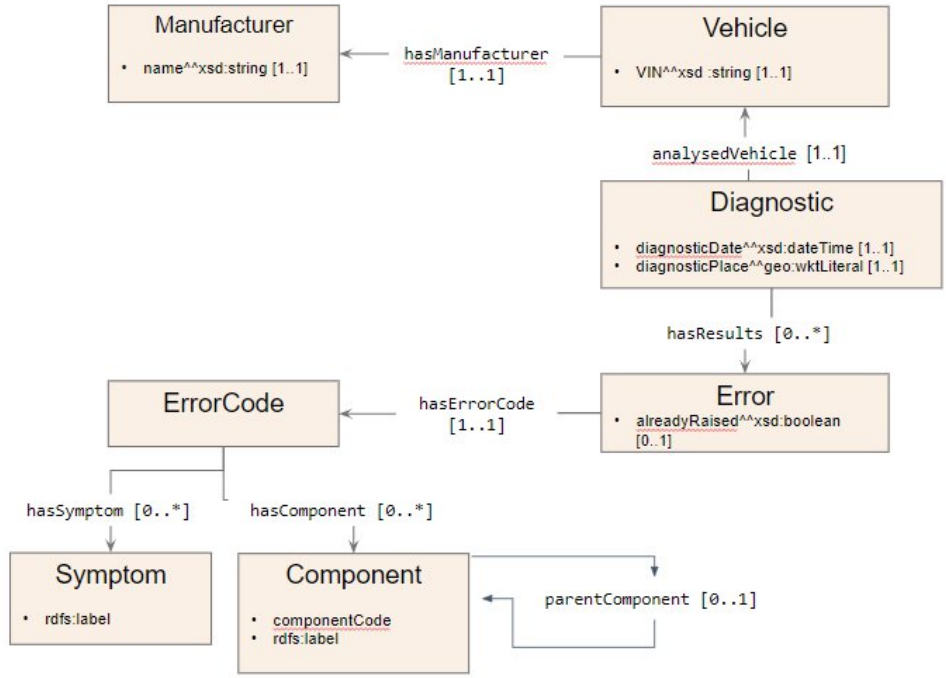
The most frequent use-case for simplifying the user view is when two classes in your data model are connected through one (or more) intermediate classes that you would like to hide in Sparnatural. For example: *“Persons live in City, and City is part of Country”*. Suppose what you would like to show to your users in the query builder is simply “*Persons live in Country”*, hiding the “City” class.

You will do this with a “*sequence path*”, by putting the two properties you want to follow using the “***/***” character. In our simple example this would be something like “<http://example.com/lives_in>/<http://example.com/is_part_of>”. This means: *“follow the lives_in property, then follow the is_part_of property”*.

Note that you can traverse more than two properties by appending the “/” character with a third property, then the “/” with a fourth, etc.

Example

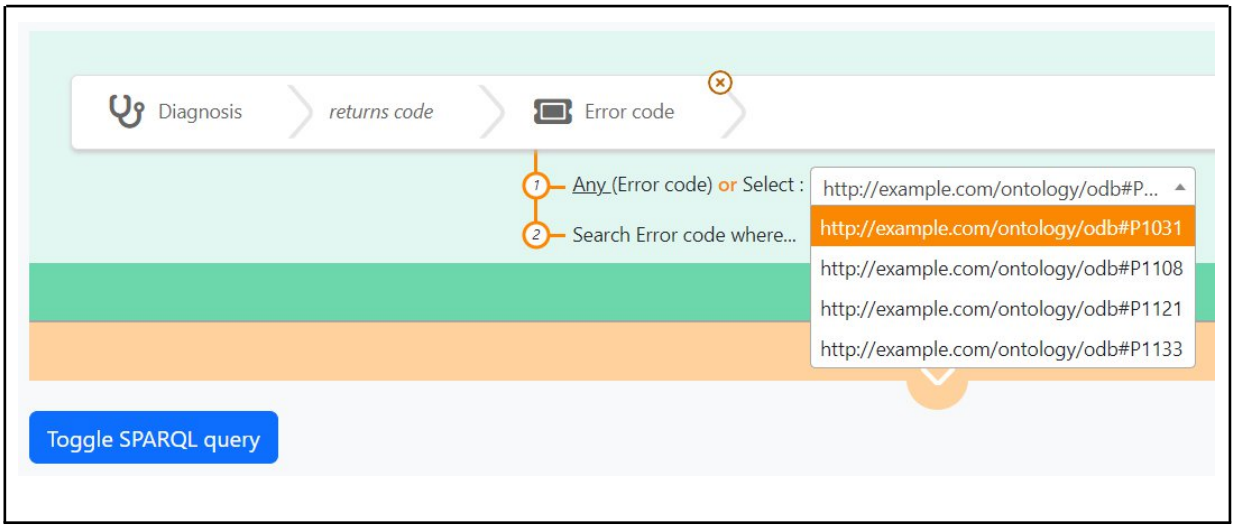
Let’s figure out, starting with the “Diagnostic” class of cars ontology, you would like to go straightly to the Error Code, going through the “Error” item that doesn’t interest you that much :



This shortcut is to be created as a new property in the configuration file, while specifying a special sequence path in the core:sparqlString column of the spreadsheet : there you need to type the exact URI of both (or as many as) properties you want to go through between brackets <>, each separated by the “/” character :

	A	C	G	H	J
1	**Ontology IRI**	<--- Don't touch this cell			
2					
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*English label of the property*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes*	*The corresponding peice of SPARQL to be inserted instead of the URI of this property. This can be a property URI enclosed in "<" ">", or a SPARQL property path. If not provided, the URI of the property is left intact in the query. The SPARQL string must NOT use prefixes.*
4	**URI**	**rdfs:label@en**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**	**core:sparqlString^^xsd:string**
11	Diagnostic				
12	odb:diagnosticDate	has diagnosis date	odb:Diagnostic	this:Attribute	
13	odb:analysedVehicle	analysed vehicle	odb:Diagnostic	odb:Vehicle	
14	odb:hasResults	has results	odb:Diagnostic	odb:Error	
15	this:returnsCode	returns code	odb:Diagnostic	odb:ErrorCode	<http://example.com/ontology/odb#hasResults>/<http://example.com/ontology/odb#hasErrorCode>
16	Error				
17	odb:alreadyRaised	already raised	odb:Error	this:Attribute	
18	odb:hasErrorCode	has error code	odb:Error	odb:ErrorCode	

Then you can see the Error item is hidden in that case in the query builder : the Error Code appears to be directly linked to the Diagnosis, so that we can directly obtain the list of Error Codes corresponding to a given Diagnosis :



Querying inverse properties

Another frequent use-case where the user view differs from the underlying graph structure is when you want to provide the user with an inverse relationship that does not exist in the data. For example if you have *"City is part of Country"* in your graph, you may want to provide the user with the ability to navigate with *"Country contains City"*.

You will do this with an *"inverse path"*, by prefixing the property URI with the "**^**" character. In our example this would be "^<http://example.com/is_part_of>". This means *"follow the is_part_of property in the inverse direction"*.

Example

In cars ontology, starting from the Vehicle, searching for a Diagnostic isn't possible if we refer to the diagram : the property goes from Diagnostic —to—> Vehicle indeed. Here we create the "this:hasDiagnosis" property, that goes from Vehicle —to—> Diagnostic, and is mapped to ^<http://example.com/ontology/odb#analysedVehicle>

	A	C	G	H	J
1	**Ontology IRI**	<--- Don't touch this cell			
2					
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*English label of the property*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes*	*The corresponding peice of SPARQL to be inserted instead of the URI of this property. This can be a property URI enclosed in "<" ">", or a SPARQL property path. If not provided, the URI of the property is left intact in the query. The SPARQL string must NOT use prefixes.*
4	**URI**	**rdfs:label@en**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**	**core:sparqlString^^xsd:string**
7	Vehicle				
8	odb:VIN	has VIN	odb:Vehicle	this:Attribute	
9	odb:hasManufacturer	has manufacturer	odb:Vehicle	odb:Manufacturer	
10	this:hasDiagnosis	has diagnosis	odb:Vehicle	odb:Diagnostic	^<http://example.com/ontology/odb#analysedVehicle>

The property now appears in the query builder note the caret “^” in the SPARQL query) :

Querying multiple properties in a single criteria

This is to be used if you would like the user to query more than one property at the same time. This can be useful if you would like to provide a search field (core:SearchProperty) that will search in label + description. This can also be used if two classes are connected by more than

one possible property and you want to search all of them, as *“Person is friend with Person”* and *“Person is a colleague of Person”*; you may want to provide your user with *“Person knows Person”*, and “knows” would search for both “is friend with” and “is colleague of”.

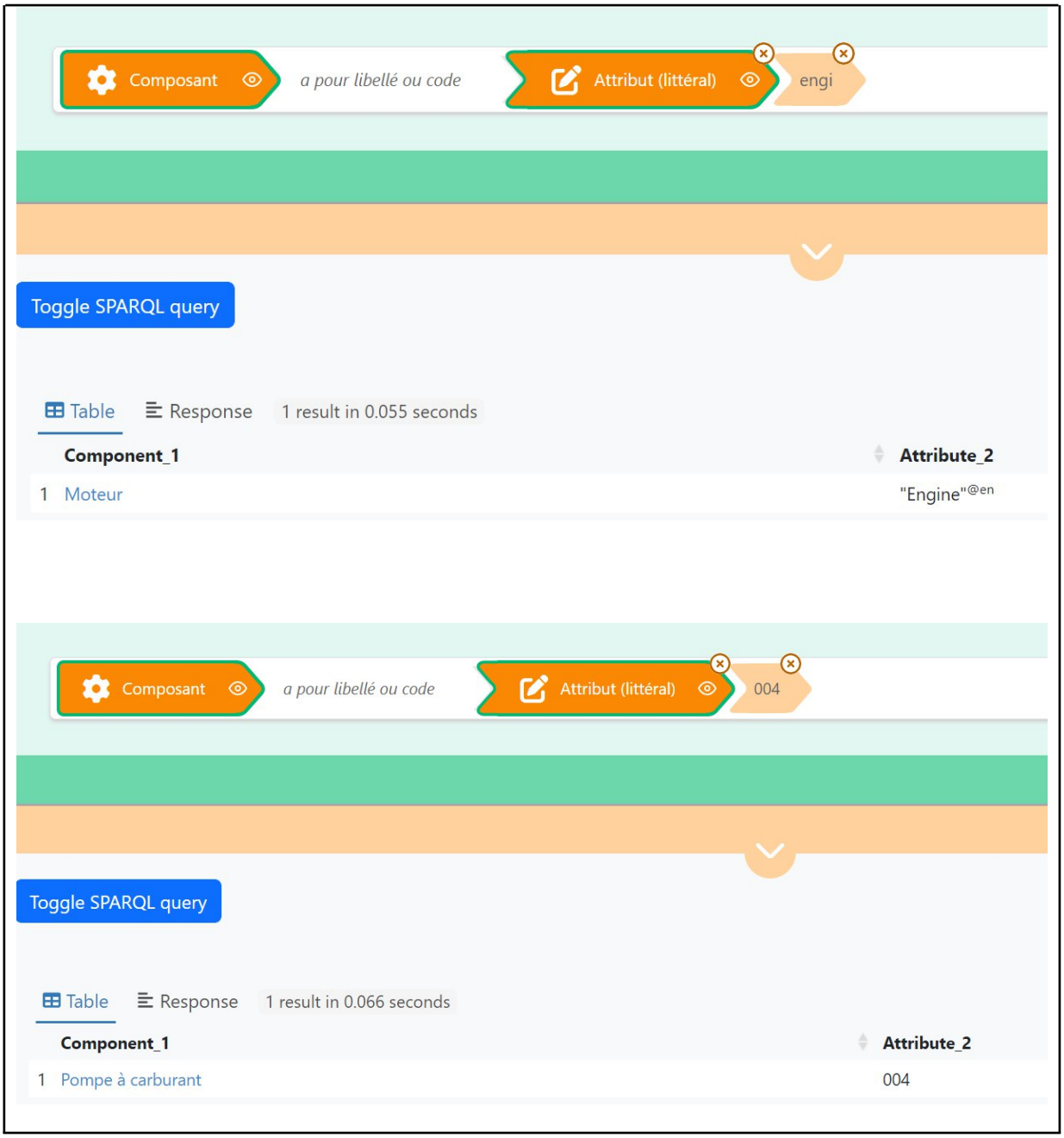
You will do this with an *“alternative path”*, by joining all properties URI with the “**|**" character. In our example this would be “<http://example.com/is_friend_of>|<http://example.com/is_colleague_of>”. This means *“follow either the is_friend_of or is_colleague_of properties”*.

Example

To illustrate this on the Component class, we decided to query both label and component code in one unique field: you can see the new property this:labelOrCode has been created therefore with a special ”|” SPARQL string to combine both properties behind a single one :

	A	C	G	H	J
1	**Ontology IRI**	<--- Don't touch this cell			
2					
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*English label of the property*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes*	*The corresponding peice of SPARQL to be inserted instead of the URI of this property. This can be a property URI enclosed in "<" ">", or a SPARQL property path. If not provided, the URI of the property is left intact in the query. The SPARQL string must NOT use prefixes.*
4	**URI**	**rdfs:label@en**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**	**core:sparqlString^^xsd:string**
24	Component				
25	odb:componentCode	has component co	odb:Component	this:Attribute	
26	rdfs:label	label	odb:Component	this:Attribute	
27	this:labelOrCode	label or code	odb:Component	this:Attribute	<http://example.com/ontology/odb#componentCode>|<http://www.w3.org/2000/01/rdf-schema#label>

We can see in the two following screenshot that a search for either a code or a label of component will work:



Querying a property recursively

This is to be used in combination with a tree property (`core:TreeProperty`). This is useful when you would like the user to query recursively and transparently into a complete "branch" of entities related with a hierarchical link (typically `skos:broader` or `dcterms:isPartOf`).
Most of the time, when you provide a tree widget, the implicit expectation from the user is that when she selects a node in the tree, then the query would also search for all children of that node.
For example if you have *"Place is part of Place"* in your graph, with places organized as a tree, if the user searches for "Restaurant located in Paris", then she would expect to receive restaurants also located in places that are part of Paris, such as "17eme arrondissement".

You will do this with a combination of “sequence path” (the “/” operator seen above) and *“zero or more path”*, by appending a “*” symbol after the property URI. In our example this would be “<http://example.com/is_located_in>/<http://example.com/is_part_of>*”. This means: *“follow the is_located_in property, then follow the is_part_of property recursively (until you reach the selected node, which in our example would be Paris)”*; In other words *“select all restaurants with a is_located_in property that points to a place that is linked to Paris with any number of is_part_of properties”*.

Combining property paths

It is possible to combine the sequence operator (“/”), inverse operator (“^”), alternative operator (“|”), and zero-or-more operator (“*”). A typical use-case is to combine inverse with a sequence operator to traverse properties in the inverse direction in a sequence path.

Example

In our “Car” ontology we could imagine a direct link between a “Vehicle” and the “Error Code” that were diagnosed on this Vehicle, which would give the property path ^<http://example.com/ontology/odb#analysedVehicle>/<http://example.com/ontology/odb#hasResults>/<http://example.com/ontology/odb#hasErrorCode>

When the same property is used on multiple classes

It may happen that the same property is used on more than one class in the data model. A typical situation is when rdfs:label is used to label many entities in the data model. In that case, and in order to keep the configuration of each entity separated from the others, it is advised to create one specific line in the “this:” namespace for each entity, and map them to the same property in the core:sparqlString column. This way, each line can be configured differently and have different labels, tooltips or widget.

For example if both foaf:Person and foaf:Organization can have the property foaf:name, you can declare this:personName with rdfs:domain foaf:Person, this:organizationName with rdfs:domain foaf:Organization, and map them both to <http://xmlns.com/foaf/0.1/name>

Tip: It is even possible to *always* use the “this:” namespace when creating the properties in the configuration, and *always* map them to an underlying property using the core:sparqlString column. This has the advantage of not mixing your ontology namespace

with the “this:” namespace in the configuration, but the disadvantage is that you need to always fill in the core:sparqlString column.

Example

In the Cars ontology, both Symptoms and Components can have rdfs:label. We chose to declare two separate lines “this:symptomLabel” and this:componentLabel, each mapped to <http://www.w3.org/2000/01/rdf-schema#label>. The label of the property (“label” in english, “a pour libellé” in French) remains the same, so it is identical from the user point of view; however tooltips can be different in each case, for example.

	A	B	C	D	H	I	K
1	**Ontology IRI**	https://data.mydomai	<--- Don't touch this cell				
2							
3	*URI of the property in the configuration. This can use prefixes declared in the first sheet*	*This must **always** be owl:ObjectProperty*	*English label of the property*	*French label of the property. Adjust the language code in the cell below if needed.*	*The reference to a class URI from the first sheet to which this property can apply. Multiple classes*	*The reference to a class URI from the first sheet that is a possible value for this property. Multiple classes*	*The corresponding peice of SPARQL to be inserted instead of the URI of this property. This can be a property URI enclosed in "<" ">", or a SPARQL property path. If not provided, the URI of the property is left intact in th query. The SPARQL string must NOT use prefixes.*
4	**URI**	**rdf:type**	**rdfs:label@en**	**rdfs:label@fr**	**rdfs:domain(separator=",")**	**rdfs:range(separator=",")**	**core:sparqlString^^xsd:string**
22	Symptom						
23	this:symptomLabel	owl:ObjectProperty	label	a pour libellé	odb:Symptom	this:Attribute	<http://www.w3.org/2000/01/rdf-schema#label>
24	Component						
25	odb:componentCode	owl:ObjectProperty	has component co	a pour code composant	odb:Component	this:Attribute	
26	this:componentLabel	owl:ObjectProperty	label	a pour libellé	odb:Component	this:Attribute	<http://www.w3.org/2000/01/rdf-schema#label>
28							

Querying a subset of a class

This is a less frequent use-case. It can be useful if your graph has very generic classes, but you want to show more specific and meaningful entries to your users. A good case is when you use SKOS Concepts, organized in different Concept Schemes.
For example if you have the class “Document” in your graph, but you want to show to the user different kinds of Documents, such as “Reports”, “Articles” or “News Item”, based on a “type” property of the Document instances.

You will do this by specifying a custom class URI in your configuration and mapping it to a SPARQL string indicating “Document with type = Report”, which would translate into “<http://example.com/Document>; <http://example.com/type> <http://example.com/Report>”

Note that this is a mapping of a class, not a property, thus to be defined in the “Classes” tab, in the “core:sparqlString” column.

Example

This example is not taken from the “Car” ontology that does not contain such a use-case.

Here, originally only the skos:Concept class is used in the graph.

Note how the class from the config “Product”, using the “this:” namespace, is aligned to all SKOS Concepts which are in the scheme Product, by means of the SPARQL string “<http://www.w3.org/2004/02/skos/core#Concept> ; <http://www.w3.org/2004/02/skos/core#inScheme> <https://data.example.org/authority/product>”

Note how the Keywords are all the Concepts that are in the scheme Thesaurus, by means of the SPARQL string “<http://www.w3.org/2004/02/skos/core#Concept> ; <http://www.w3.org/2004/02/skos/core#inScheme> <https://data.example.org/authority/thesaurus>“

	URI	rdfs:label@en	config-core:sparqlString^^xsd:string	config-datasources:datasource
28	this:Product	Product	<http://www.w3.org/2004/02/skos/core#Concept> ; <http://www.w3.org/2004/02/skos/core#inScheme> <https://data.example.org/authority/product>	
29	this:Keyword	Keyword (Thesaurus)	<http://www.w3.org/2004/02/skos/core#Concept> ; <http://www.w3.org/2004/02/skos/core#inScheme> <https://data.example.org/authority/thesaurus>	

Querying more than one class

This is a less frequent use-case. It can be useful if your graph has specific classes, but you want to show more generic entries to your users.
For example if you have the classes “Person” and “Company”, but you want to show to the user a single entry like “Actors”, encompassing both persons and companies.

You will do this by specifying a custom class URI in your configuration and mapping it to a SPARQL string indicating “Person or Company”, which would translate into “?type VALUES ?type { <http://Person> <http://Company>}” .

Note that this is a mapping of a class, not a property, thus to be defined in the “Classes” tab, in the “core:sparqlString” column.

Create a Multilingual configuration

Sparnatural is multilingual by nature and can display the labels and tooltips from its configuration in multiple languages, if they are provided in the configuration. The “<spar-natural>” HTML element contains a “lang” attribute that indicates which language should be used to select the labels and tooltips to display[2]. That attribute can be adjusted by a control in

[2] Note however that the few hardcoded labels of Sparnatural exist in French and English only.

the HTML page (out of scope of Sparnatural and of this documentation), typically a language-selection dropdown.

If you want to provide your users with a multilingual configuration you have to add additional columns in your configuration files:

- In the “Classes” tab:
  - add more “rdfs:label@xx” columns and adjust the language tag in the header to populate the labels of classes in different languages
  - add more “core:tooltip@xx” columns and adjust the language tag in the header to populate the tooltips of classes in different languages
- In the “Properties” tab, duplicate the same columns “rdfs:label@xx” and “core:tooltip@xx” for the labels and tooltips of the properties.

Advanced note: Sparnatural is also configured with a “defaultLang” parameter. This default language is the language in which the knowledge graph is supposed to always have a label for all entities. This is meant to deal with situations where some entities do have a label in the user preferred language, and others don’t, but will have a label in the default language. The default label can be returned to display a label to the user.

Example

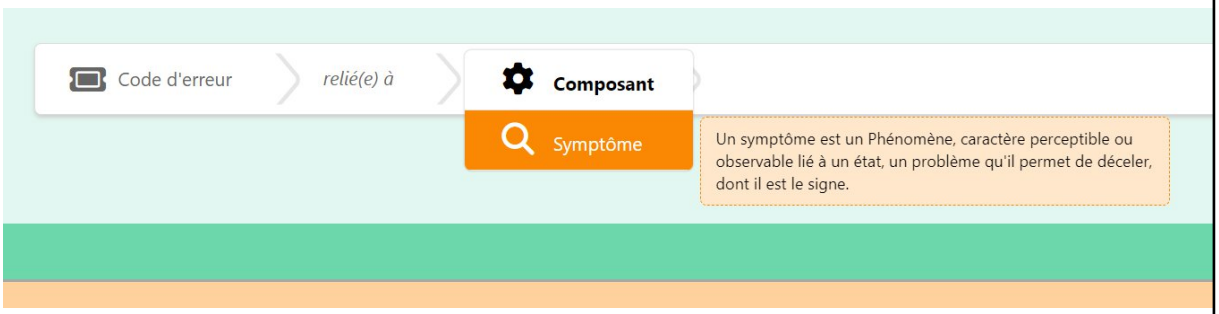
Classes and properties labels and tooltips can be translated in as many languages as wished just by adding the translations in an “@xx” column for each : here the classes tab, translated in French, `rdfs:label@fr` and `core:tooltip@fr` :

	URI	rdf:type	rdfs:subClassOf	rdfs:label@en	rdfs:label@fr	core:falcon	core:tooltip@en	core:tooltip@fr
12	*URI of the class. This column can use prefixes declared above in the header*	*This should **always** be owl:Class*	*This should be set to core:SparnaturalClass most of the time, or rdfs:Literal when it corresponds to a literal value*	*English label of the class*	*French label of the class. Adjust the language code in the cell below to another language if needed.*	*The Fontawesome icon code for the class, e.g. "fa-duotone fa-user". Search for icon codes at https://fontawesome.com/. Fontawesome provides a limited number of icons for free, and you can buy a license to access the full set of icons.*	*The english tooltip for the class.*	*The french tooltip of the class. Adjust the language code in the cell below to another language if needed.*
13								
14	odb:Manufacturer	owl:Class	core:SparnaturalClass	Manufacturer	Constructeur	fa-solid fa-industry	A car manufacturer is a company	Un constructeur automobile est
15	odb:Vehicle	owl:Class	core:SparnaturalClass	Vehicle	Véhicule	fa-solid fa-car	A vehicle is a car model for a spec	Un véhicule est un modèle de v
16	odb:Diagnostic	owl:Class	core:SparnaturalClass	Diagnosis	Diagnostic	fa-solid fa-stethoscope	A diagnosis identifies a possible p	Un diagnostic permet d'identifie
17	odb:Error	owl:Class	core:SparnaturalClass	Error	Erreur	fa-solid fa-circle-exclamation	An error is an element that comes	Une erreur est un élément qui r
18	odb:ErrorCode	owl:Class	core:SparnaturalClass	Error code	Code d'erreur	fa-solid fa-ticket	An error code is a set of numbers	Un code erreur, est un ensemb
19	odb:Symptom	owl:Class	core:SparnaturalClass	Symptom	Symptôme	fa-solid fa-magnifying-glass	A symptom is a phenomenon, per	Un symptôme est un Phénomèn
20	odb:Component	owl:Class	core:SparnaturalClass	Component	Composant	fa-solid fa-gear	A class representing a component	Une classe représentant un cor
21								
22	this:Attribute	owl:Class	rdfs:Literal	Attribute	Attribut (littéral)	fa-solid fa-pen-to-square	A class to display literal values (as	Une classe pour afficher les val

here the properties one, `rdfs:label@fr` and `core:tooltip@fr` again :

	A	C	D	E	F	G
1	**Ontology IRI** ai	<--- Don't touch this cell				
2						
	URI of the property in the configuration. This can use prefixes declared in the first sheet	*English label of the property*	*French label of the property. Adjust the language code in the cell below if needed.*	*Indicates the widget type of the property. This can take its value in one of the predefined sparnatural property types.*	*The english tooltip for the property.*	*The english tooltip for the property.*
3						
	URI	**rdfs:label@en**	**rdfs:label@fr**	**rdfs:subPropertyOf**	**core:tooltip@en**	**core:tooltip@fr**
4						
5	Manufacturer					
6	odb:name	has name	nom	core:NonSelectableProperty	Specifies the name of the manufacturer.	Spécifie le nom du constructeur.
7	Vehicle					
8	odb:VIN	has VIN	a pour VIN	core:AutocompleteProperty	Specifies the Vehicle Identification Number (VIN) of the vehicle.	Spécifie le numéro d'identification du véhicule (VIN).
9	odb:hasManufacturer	has manufacturer	a pour constructeur	core:ListProperty	Specifies the manufacturer of the vehicle.	Spécifie le constructeur d'un véhicule.
10	this:hasDiagnosis	has diagnosis	a pour diagnostic	core:NonSelectableProperty	The property is the inverse of odb:analysedVehicle.	Propriété inverse de odb:analysedVehicle.
11	Diagnostic					
12	odb:diagnosticDate	has diagnosis date	date du diagnostic	core:TimeProperty-Date	Defines the date on which the diagnosis occurs.	Définit la date à laquelle le diagnostic a eu lieu.
13	odb:analysedVehicle	analysed vehicle	véhicule analysé	core:AutocompleteProperty	Specifies that the vehicle has been analyzed, to identify a potential problem.	Spécifie que le véhicule a été analysé, pour identifier un potentiel problèm
14	odb:hasResults	has results	a pour résultat	core:NonSelectableProperty	Specifies the results, from the analysis.	Spécifie les résultats issus de l'analyse.
15	this:returnsCode	returns code	renvoie le code	core:ListProperty	The property is a shortcut between Diagnosis and Error Code.	Cette propriété est un raccourci entre Diagnostic et Code d'erreur.
16	Error					
17	odb:alreadyRaised	already raised	déjà signalée	core:BooleanProperty	Attribute indicating whether an error has already been detected previously.	Attribut permettant de savoir si une erreur a déjà été relevée précédemm
18	odb:hasErrorCode	has error code	a pour code d'erreur	core:ListProperty	Specifies the error code relating to an error reported during a diagnostic.	Spécifie le code erreur relatif à une erreur remontée lors d'un diagnostic.
19	ErrorCode					
20	odb:hasSymptom	has symptom	a pour symptôme	core:ListProperty	Specifies the symptoms associated with an error code.	Spécifie le symptôme associé à un code erreur.
21	odb:hasComponent	has component	concerne le composant	core:ListProperty	Specifies the components impacted by an error code.	Spécifie le composant impacté par un code erreur.
22	Symptom					
23	rdfs:label	label	a pour libellé	core:SearchProperty	Specifies the name of the object.	Spécifie le nom de l'objet.
24	Component					
25	odb:componentCode	has component co	a pour code composant	core:SearchProperty	Specifies the unique code of the component.	Spécifie le code unique relatif à un composant.
26	rdfs:label	label	a pour libellé	core:SearchProperty	Specifies the name of the object.	Spécifie le nom de l'objet.
	this:labelOrCode	label or code	a pour libellé ou code	core:SearchProperty	Allows to get a label or a code.	Permet de rechercher un libellé ou un code.
27						

This makes it possible to have a Sparnatural interface in French, by adjusting the “src” attribute of the <spar-natural> element in the HTML page to “fr”:



Displaying labels in the result table

Default label properties

By default, when triggering a query, you will get a list of URIs as result. URIs are not very nice to display for users, who will want to see a clickable human-readable label instead. Sparnatural allows to indicate what is the label property to use when running the query and displaying the results in the table. To do this, populate the “core:defaultLabelProperty” column in the “Classes” tab, with the URI of one of the properties from the “Properties” tab. This property then becomes the default label property of this class and will be automatically fetched whenever this class is selected as a column in the result set, with the “eye” icon of an orange arrow.

The property you refer to can be any property from the Properties tab. In practice it will usually correspond to a property that has in its range a Class that is indicated as an rdfs:subClassOf of rdfs:Literal because it is a Literal property. Typical default label properties correspond to rdfs:label, foaf:name, skos:prefLabel, etc.

The property you refer to can use the "this:" namespace and be mapped to an underlying SPARQL property path in its core:sparqlString column.

Concretely, this means the following: when selecting an entity from the query builder, for example "Person", Sparnatural will generate a variable "?Person_4". If the "Person" class is annotated with "core:defaultLabelProperty" that points to a property in your configuration, Sparnatural will automatically return the variable "?Person_4_label" populated with the property.

Advanced note: you can mark the default label property as optional, with core:enableOptional. Sparnatural will honor this by always returning the xxxx_label in the query and populating it only when it is known (as opposed to not returning the row if the property is missing on an item).

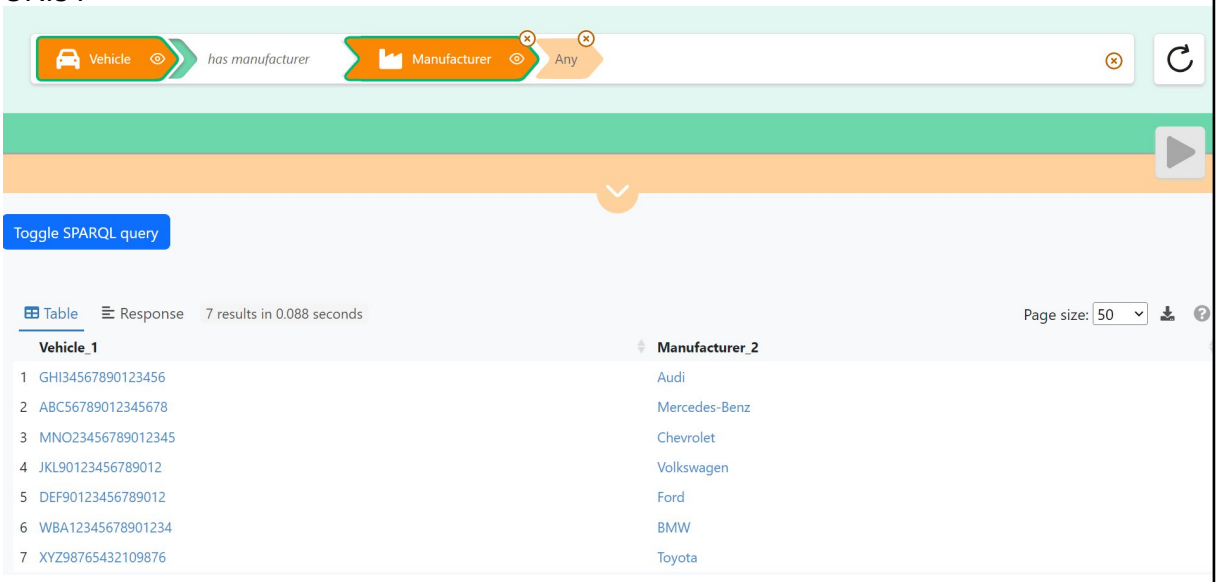
Tip: sometimes the default label property for a class is available to the user as a property that can be searched on. For example Persons might have "name" as their default label property, and you want the user to search on person names with an autocomplete widget. But sometimes you want the default label property to be hidden in the query builder, and you simply need it to be fetched in the result table. In that case, proceed exactly as normal, except that you don't set an rdfs:domain on the property used as the default label property. Leave the rdfs:domain column empty for that property. Properties without domain are still part of the configuration but hidden in the query builder.

Example

In this case we decided to display the Manufacturer's names by using the odb:name property as a default label, the VIN number for the Vehicles (odb:VIN), the this:symptomLabel for Symptoms and the this:componentLabel for the Components. This is specified in the core:defaultLabelProperty column :

	URI	rdf:type	rdfs:subClassOf	rdfs:label@en	core:defaultLabelProperty
13					
14	odb:Manufacturer	owl:Class	core:SparnaturalClass	Manufacturer	odb:name
15	odb:Vehicle	owl:Class	core:SparnaturalClass	Vehicle	odb:VIN
16	odb:Diagnostic	owl:Class	core:SparnaturalClass	Diagnosis	
17	odb:Error	owl:Class	core:SparnaturalClass	Error	
18	odb:ErrorCode	owl:Class	core:SparnaturalClass	Error code	
19	odb:Symptom	owl:Class	core:SparnaturalClass	Symptom	this:symptomLabel
20	odb:Component	owl:Class	core:SparnaturalClass	Component	this:componentLabel

The result in the query builder is much more explicit and user-friendly than simple plain URIs !



Multilingual default label properties

By default, when fetching the default label property, Sparnatural will not apply any language filter; so multiple values will be retrieved in case the label property holds multilingual values. In order to instruct Sparnatural to retrieve the default label property only in the current user language, set the core:isMultilingual column of that property to true.

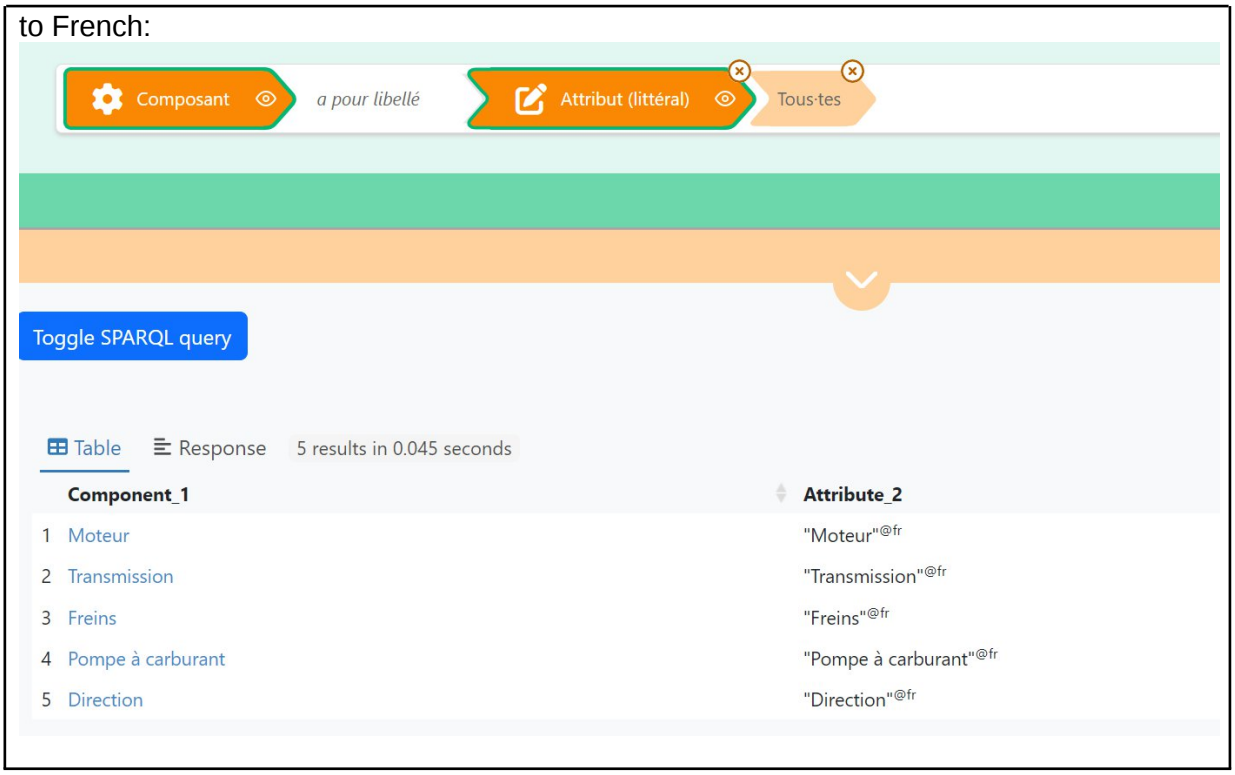
Example

In the example data of the cars ontology, labels of components are multilingual, e.g. “Engine”@en and “Moteur”@fr. They are declared in the this:componentLabel configuration property. In order to indicate to Sparnatural that only the label in the current user language should be retrieved, we set “TRUE” in the core:isMultilingual column:

	URI	rdfs:label@en	rdfs:label@fr	rdfs:domain(separator=",")	rdfs:range(separator=",")	core:isMultilingual^^xsd:boolean
4						
23	this:symptomLabel	label	a pour libellé	odb:Symptom	this:Attribute	
24	Component					
25	odb:componentCode	has component code	a pour code composant	odb:Component	this:Attribute	
26	this:componentLabel	label	a pour libellé	odb:Component	this:Attribute	VRAI
27	this:labelOrCode	label or code	a pour libellé ou code	odb:Component	this:Attribute	

We can see that only French labels are retrieved in the result table, when Sparnatural is set

to French:



Advanced configuration

Advanced configuration : creating custom datasources

Creating a custom datasource to populate a list property or an autocomplete property is possible by providing your custom SPARQL query. To do this you need to be proficient with SPARQL.

To create your custom datasource, go to the “Datasources” tab of the configuration file, and:

- Add a line, with your datasource URI in column A, in the “this:” namespace
- in column rdf:type, set the value `datasources:SparqlDatasource`
- in column datasources:queryString, enter the SPARQL query, including all its prefixes.
- then you can refer to your datasource from the “datasources:datasource” column of the “Properties” tab.

The datasources documentation explains the rules you need to follow to create your own SPARQL datasource. Please refer to this documentation for details. To sum it up, your query:

- must return 2 variables ?uri and ?label
- can take advantage of special variables that will be passed by Sparnatural before the query is sent, such as $domain with the class selected at the beginning of the criteria, $range with the class selected at the end, $property with the property selected, $lang with current user language, etc. You don’t *have to* use all of them.

If you don’t see any results in your dropdown list populated with a custom query, refer to the next section to know how to debug the query.

Example

Here we propose to set a custom datasource for `odb:hasComponent` property. Let’s imagine it would be created using a concatenation of component code + component label. To do so we first write the SPARQL query that will be sent to the system to get the info, then we can embed it in a new “this” datasource (tab “Datasources” of Sparnatural config sheet) :

3	*URI of the datasource in the configuration. This is the value that should be referenced from the "datasources:datasource" column in the properties tab*	*This must **always** be datasources:SparqlDatasource*	*Contains the query string, containing specific Sparnatural variables. See http://docs.sparnatural.eu/OWL-based-configuration-datasources.html#your-own-sparql-query-lists--autocomplete*
4	**URI**	**rdf:type**	**datasources:queryString**
5	this:list_myname_count	datasources:SparqlDatasource	
6	this:list_skosprefLabel_alpha_with_count_langfr	datasources:SparqlDatasource	
7	this:list_odbname_alpha	datasources:SparqlDatasource	
8	this:search_VIN_strstarts	datasources:SparqlDatasource	

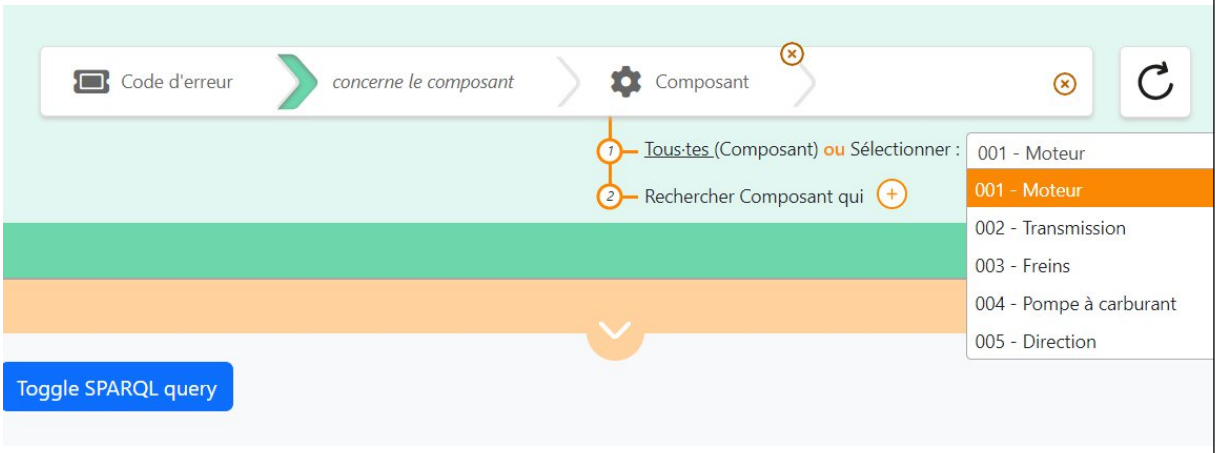
	this:list_componentCode_alpha	datasources:SparqlDatasource	PREFIX odb: <http://example.com/ontology/odb#>
			SELECT DISTINCT ?uri ?label
			WHERE {
			?domain $type $domain .
			?domain $property ?uri .
			# Note how the range criteria is not used in this query
			FILTER(isIRI(?uri))
			?uri rdfs:label ?libelleComposant .
			FILTER(lang(?libelleComposant) = "" || lang(?libelleComposant) = $lang)
			?uri odb:componentCode ?codeComposant .
			# Concat component code + component label
			BIND(CONCAT(STR(?codeComposant)," - ",STR(?libelleComposant)) AS ?label)
			}
			ORDER BY UCASE(?label)
			LIMIT 500

The details of the SPARQL query is beyond the scope of this documentation, please simply note that a/ it is using “magic variables” $domain, $property, $lang that are replaced at runtime by Sparnatural with the corresponding values in the criteria being built (see the Sparnatural datasource documentation) and b/ note the `BIND(CONCAT(...) AS ?label)` line that is doing the actual concatenation of the code with the name, which is returned in the result set.

Next step is modifying property’s datasource itself with the URI of the new datasource :

	URI	rdfs:label@en	rdfs:domain(separator=",")	rdfs:range(separator=",")	datasources:datasource
4					
5	Manufacturer				
6	odb:name	has name	odb:Manufacturer	this:Attribute	
7	Vehicle				
8	odb:VIN	has VIN	odb:Vehicle	this:Attribute	
9	odb:hasManufacturer	has manufacturer	odb:Vehicle	odb:Manufacturer	this:list_odbname_alpha
10	this:hasDiagnosis	has diagnosis	odb:Vehicle	odb:Diagnostic	
11	Diagnostic				
12	odb:diagnosticDate	has diagnosis date	odb:Diagnostic	this:Attribute	
13	odb:analysedVehicle	analysed vehicle	odb:Diagnostic	odb:Vehicle	this:search_VIN_strstarts
14	odb:hasResults	has results	odb:Diagnostic	odb:Error	
15	this:returnsCode	returns code	odb:Diagnostic	odb:ErrorCode	
16	Error				
17	odb:alreadyRaised	already raised	odb:Error	this:Attribute	
18	odb:hasErrorCode	has error code	odb:Error	odb:ErrorCode	
19	ErrorCode				
20	odb:hasSymptom	has symptom	odb:ErrorCode	odb:Symptom	datasources:list_rdfslabel_alpha
21	odb:hasComponent	has component	odb:ErrorCode	odb:Component	this:list_componentCode_alpha
22	Symptom				
23	this:symptomLabel	label	odb:Symptom	this:Attribute	
24	Component				
25	odb:componentCode	has component code	odb:Component	this:Attribute	
26	this:componentLabel	label	odb:Component	this:Attribute	
	this:labelOrCode	label or code	odb:Component	this:Attribute	
27					

Then testing the query in the query builder to check that the query works well :

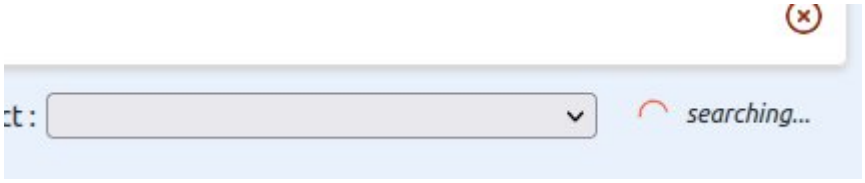


Advanced configuration : debugging custom datasources

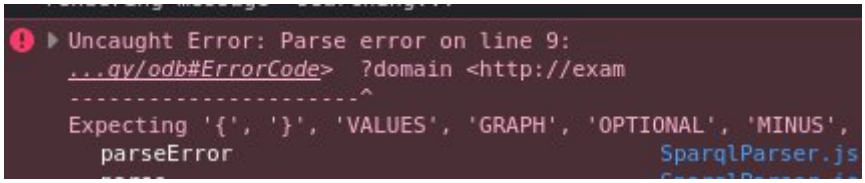
Most of the time a custom datasource query will not work the first time and a little debugging is necessary. There are three main reasons a custom datasource is not working:

Case 1 : the SPARQL query is syntactically wrong

UI Symptom : the loader keeps running, the list is not populated.



Console Symptom : Check in your console to see if there is a SPARQL parsing error message, like so:

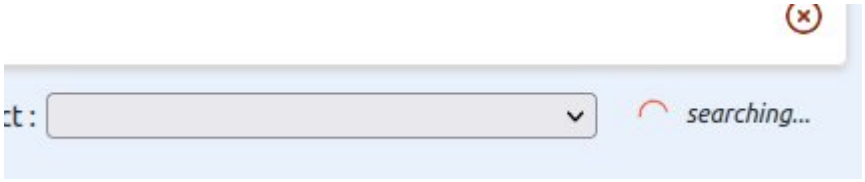


(in our case here, a missing dot in the SPARQL).

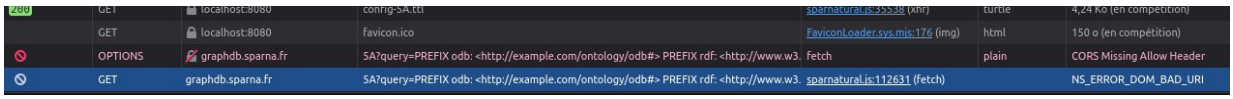
How to fix it : fix your SPARQL query, make sure you edit it in a tool with syntax checking.

Case 2 : The query to the endpoint failed (the server is unreachable, or there is a CORS issue, etc.)

UI Symptom : the loader keeps running, the list is not populated.



Console Symptom : you will see a network query failing in the network console:

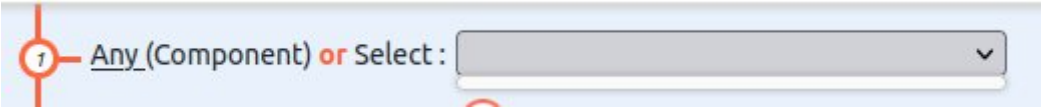


(in our case here, we simulated a CORS issue).

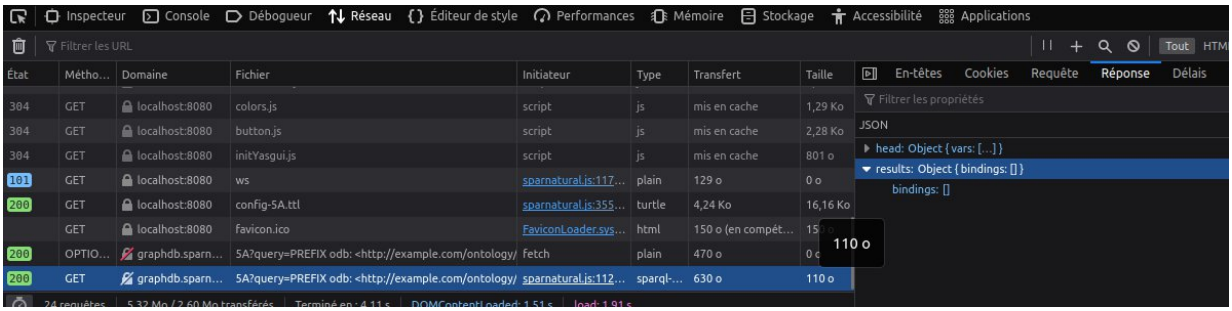
How to fix it : check more in detail why the network call failed. This could be for a security reason, a CORS reason, or another reason on the server that would return an HTTP 500 error.

Case 3 : The SPARQL query is syntactically correct and was successfully executed, but returned no results.

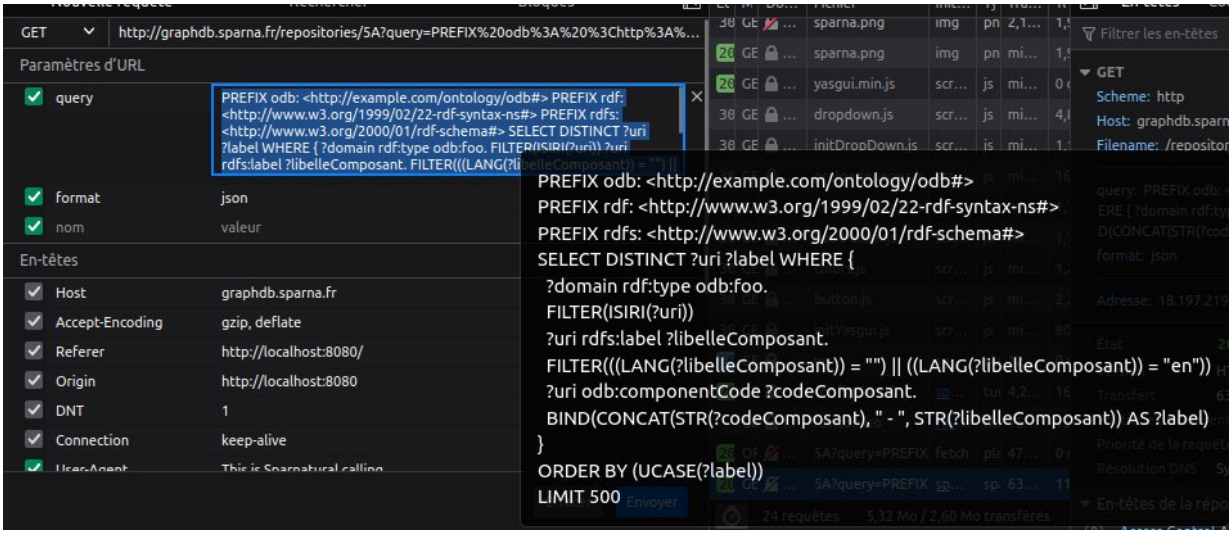
UI Symptom : the loader stops, the list is empty



Console Symptom : you will see the SPARQL HTTP request to populate the list was sent and was successful, but has returned no “bindings” in its response



How to fix it : You must understand why the query does not return the expected result. To do that you need to fetch it from the HTTP request in the console:



Copy the query, paste it in your triplestore SPARQL interface, and work on it to understand why it does not return the expected results.

> **Warning** : remember that this is the final query being sent, after all “magic variables” have been replaced by Sparnatural with their final values. Please refer to the datasource documentation for explanations on these variables. When you understand why the query does not work, remember to replace all fixed variables back with their magic variable name (e.g. $domain, $lang, etc.)

Advanced configuration : setup tree widget datasource

A tree widget requires two datasources : one to get the roots of the tree, and one to get the children of a node that is unfolded. This is set with the datasources:treeRootsDatasource and datasources:treeChildrenDatasource columns respectively, in the “Properties” tab. These two columns are useful only when the property is a core:TreeProperty, you can ignore them otherwise. The datasource documentation gives the details of the existing default tree datasources and how to create a new tree widget datasource. Please refer to this documentation for details.

Example

In Sparnatural car configuration, the class odb:ErrorCode has a property odb:hasComponent, which refer to car components that re structured in a hierarchized manner. Therefore we can set this property as a core:TreeProperty with two custom tree datasources, one identified with this:tree_root_Component and one identified with this:tree_children_Component, which serve respectively to fetch the roots and the children of a node.

4	URI	rdf:type	datasources:queryString

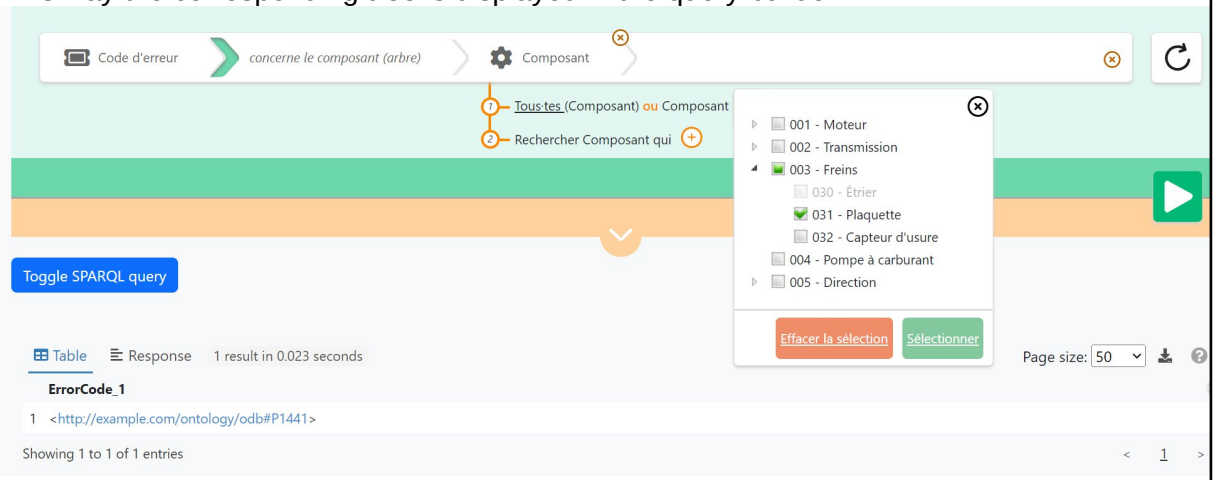
10	this:tree_root_Component	datasources:SparqlDatasource	PREFIX odb: <http://example.com/ontology/odb#>
			PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#label>
			SELECT ?uri ?label ?hasChildren (COUNT(?x) AS ?count) WHERE {
			?uri a odb:Component .
			# Keep only roots, that do not have any parent
			FILTER NOT EXISTS {
			?uri odb:parentComponent ?parent .
			}
			?uri rdfs:label ?libelleComposant .
			FILTER(lang(?libelleComposant) = "" || lang(?libelleComposant) = $lang)
			?uri odb:componentCode ?codeComposant .
			# Concat component code + component label
			BIND(CONCAT(STR(?codeComposant)," - ",STR(?libelleComposant)) AS ?label)
			OPTIONAL { ?uri ^odb:parentComponent ?children }
			BIND(IF(bound(?children),true,false) AS ?hasChildren)
			OPTIONAL {
			?x a $domain .
			?x $property ?uri .
			}
			}
			GROUP BY ?uri ?label ?hasChildren
			ORDER BY ?label

11	this:tree_children_Component	datasources:SparqlDatasource	PREFIX odb: <http://example.com/ontology/odb#>
			PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#label>
			SELECT DISTINCT ?uri ?label ?hasChildren (COUNT(?x) AS ?count) WHERE {
			
			$node ^odb:parentComponent ?uri .
			
			?uri rdfs:label ?libelleComposant .
			FILTER(lang(?libelleComposant) = "" || lang(?libelleComposant) = $lang)
			?uri odb:componentCode ?codeComposant .
			# Concat component code + component label
			BIND(CONCAT(STR(?codeComposant)," - ",STR(?libelleComposant)) AS ?label)
			
			OPTIONAL { ?uri ^odb:parentComponent ?children }
			BIND(IF(bound(?children),true,false) AS ?hasChildren)
			
			OPTIONAL {
			?x a $domain .
			?x $property ?uri .
			}
			}
			GROUP BY ?uri ?label ?hasChildren
			ORDER BY ?label

Selecting the core:TreeProperty widget from properties tab, these two datasources are then referred to like so :

4	URI	core:isMultilingual^^xsd:boolean	datasources:treeRootsDatasource	datasources:treeChildrenDatasourc
17	Error			
18	odb:alreadyRaised			
19	odb:hasErrorCode			
20	ErrorCode			
21	odb:hasSymptom			
22	this:hasComponentList			
23	this:hasComponentTree		this:tree_root_Component	this:tree_children_Component
24	Symptom			
25	this:symptomLabel			
26	Component			
27	odb:componentCode			
28	this:componentLabel	VRAI		
29	this:labelOrCode			

This way the corresponding tree is displayed in the query builder :



Note how 1/ some items in the component tree are greyed out because no error codes affect them and 2/ some items in the component tree cannot be unfolded as they have no children. Those two informations (the fact that a node has children and the fact it is not referenced as a value) are computed by the SPARQL queries used as datasources, respectively in the ?hasChildren variable and the ?count variable.

The result listed is the only error code affecting the component selected in the tree up above.