Protégé Product Tutorial

Introduction

In this tutorial you will learn how to create an ontology using the open-source ontology editor. Ontologies are used to capture knowledge about some domain of interest. An ontology describes the concepts in the domain and also the relationships that hold between those concepts.

Getting started

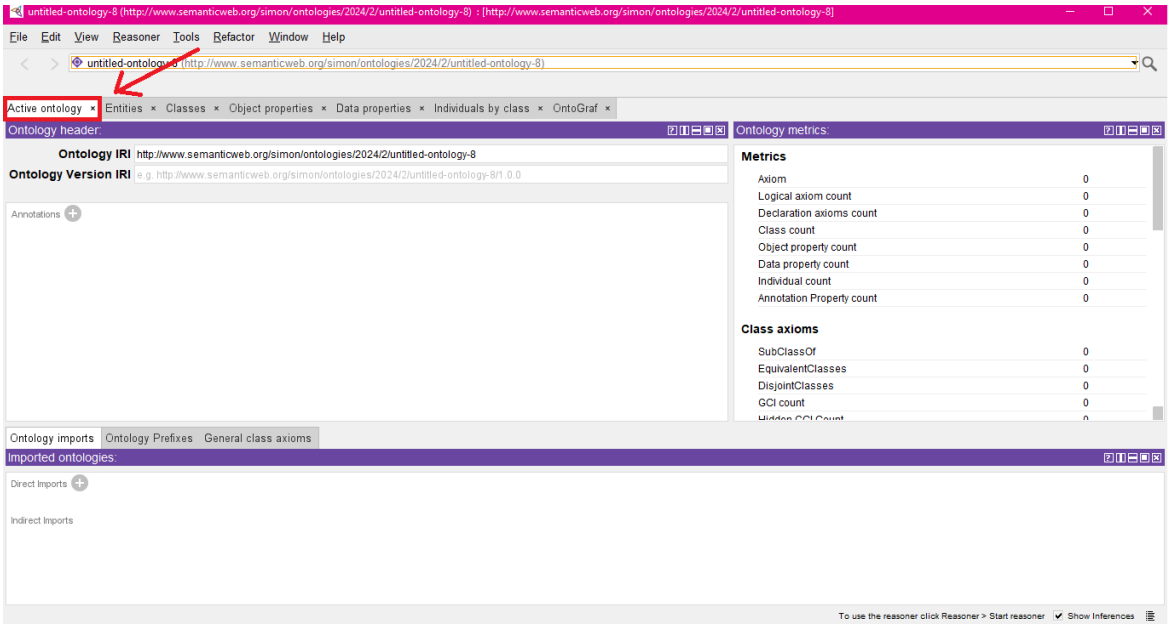
First we want to install Protégé Desktop from the official website:

https://protege.stanford.edu/software.php

Follow the instructions of the installer and then open the application after the installation has concluded.

1. Create a new OWL Ontology

When Protégé opens for the first time each day it puts up a screen of all the available plugins. You can also bring this up at any time by using File>Check for plugins. You won't need any plugins at this point of the tutorial so just click the "Not now" button.



The Protégé user-interface consists of several tabs such as *Active ontology*, Entities, etc. When you start Protégé you should be in the Active Ontology tab. This is for overview information about the entire ontology. Protégé always opens with a new untitled ontology you can start with.

On the bottom, you can optionally import other ontologies, which might be beneficial if you want your ontology to be based on an existing top-level ontology, or if you want to integrate vocabulary about a domain, like measurements.
On the right you see some ontology metrics.

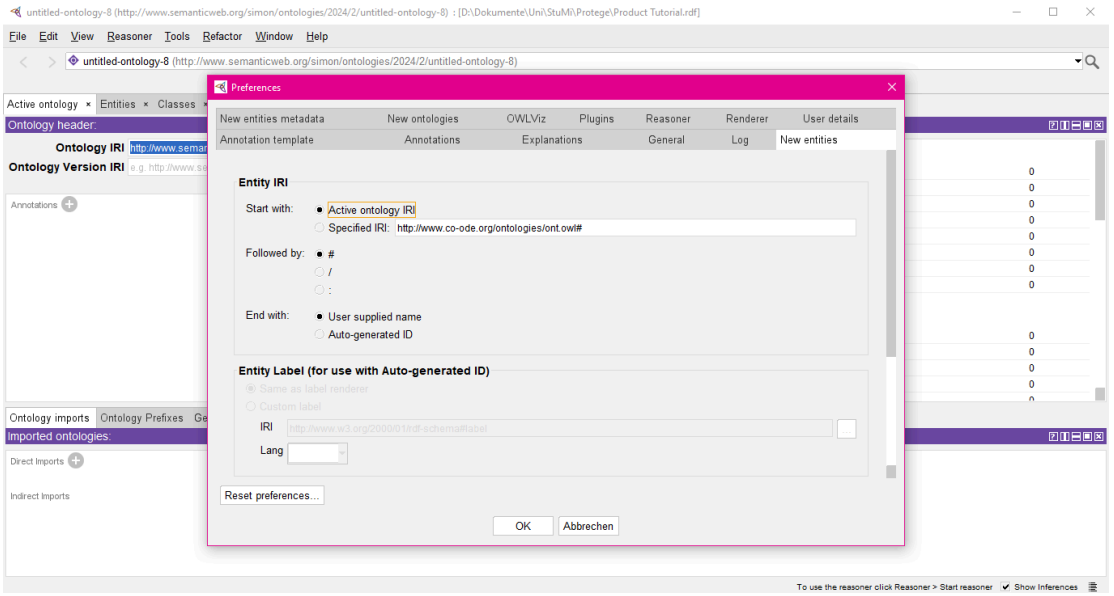
At start, your ontology should have an automated *name (an IRI*), which is something like this: "http://www.semanticweb.org/simon/ontologies/2024/2/untitled-ontology-8". You can edit the name of the ontology to something like "ProductTutorial", e.g. "http://www.mysite.de/ProductTutorial". An IRI does not have to really exist like an URL since it is an identifier that can also be assigned to named entities, like your ontology. If you want, you can also create a folder on your personal webpage where you upload your ontology, and use that URL as your ontology name.
Another option is to use purl identifier. Here, you assign a purl URL that redirects to any storage location of your ontology (see https://purl.prod.archive.org/ for more on this).

Now you can save your new ontology. Select File>Save. This should bring up a window that says: Choose a format to use when saving the "ProductTutorial" ontology. There is a drop down menu of formats to use. The default RDF/XML Syntax should be selected by clicking the "OK" button. The RDF/XML syntax is sufficient for most use cases. The different formats have different pros and cons. Some find turtle to be better to read (when opening the file in an editor). Clicking on the button should bring up the standard dialog your operating system uses for saving files. Navigate to the folder you want to use and then type in the file name, something like "Product Tutorial" and select Save.

Set the Preferences for New Entities and Rendering

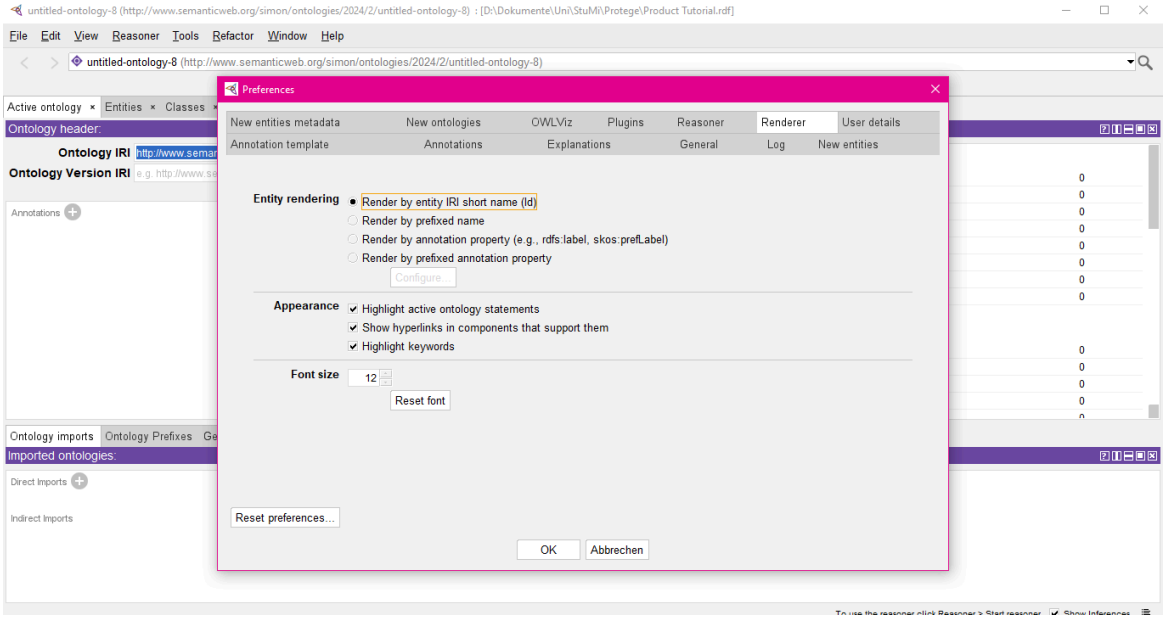
This step is **optional**, to set some preferences related to the names of new entities.
Remember that in an Ontology subject, predicate and object, i.e. any class, individual, object property, data property, annotation property, or rule is referred to as an entity.
The display name in Protègè can actually refer to two different concepts. It can be the last part of the IRI (that means anything after the last /) or it can refer to the annotation property (usually rdfs:label) used to provide a more user-friendly name for the entity.
For now, we just want to set the parameters so that in the future (especially when we do SPARQL queries) everything will work correctly.

1. Go to File>Preferences in Protégé. This will bring up a window with a few different tabs. Click on the "*New Entities" tab*. Here we want to make sure, that the following settings are enabled:
   - Start with: Active Ontology IRI
   - Followed by: #
   - End with: user supplied name



2 . Now select the *“Renderer” tab* . Here we want to make sure that these settings are selected:

- Entity rendering: Render by entity IRI short name (id)
- Appearance: check all three

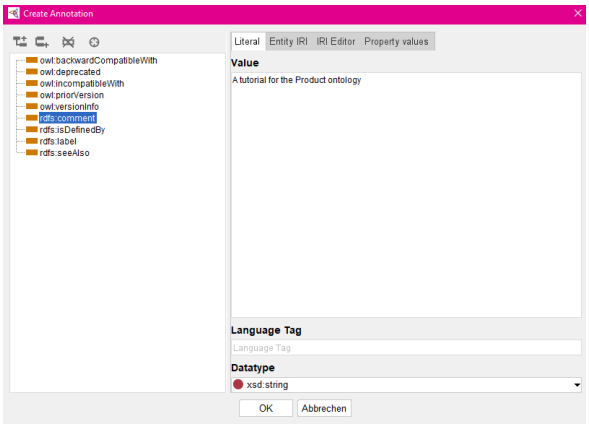


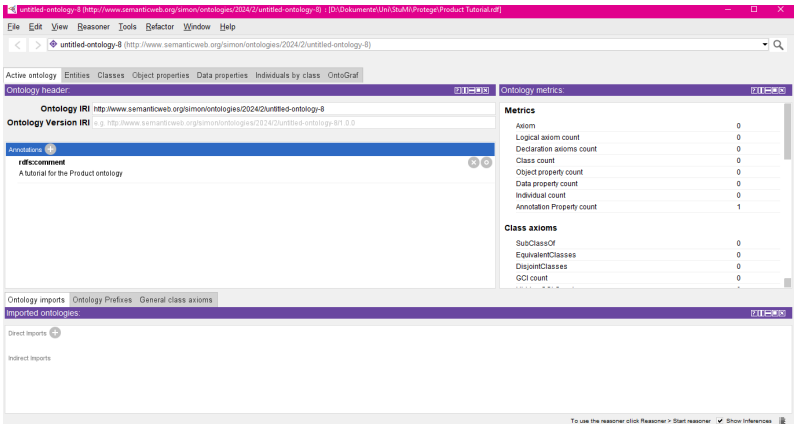
Add a Comment Annotation to Your Ontology

Now let us describe the created ontology so users get an idea what it is about.
Make sure you are in the Active Ontology tab. In the view just below the Ontology IRI and Ontology Version IRI fields, find the Annotations option and click on the “+” sign. This will bring up a menu to create a new annotation for the ontology.

The r*dfs:comment annotation* should be highlighted by default. If it isn’t highlighted, select it. Then type a new comment into the view to the right. Something like: “A tutorial for the Product ontology” and click “OK”. Note that you can set the type of the entered data as well

as a language label. When a user has opted to show entities based on annotations, they will see the labels in their system language, in case different language labels are available.





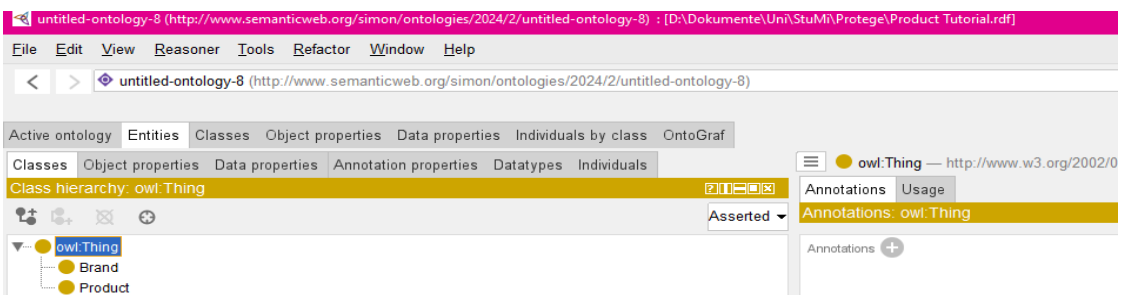
2. Create classes: Product und Brand

The main building blocks of an ontology are classes and their relations. In Protégé, editing of classes can be done in the *Entities tab*. The Entities tab has a number of sub-tabs. When you select it, the default should be the Class hierarchy view. All empty ontologies contain one class called owl:Thing. The class owl:Thing is the parent class of any ontology. It represents a set containing all individuals. Because of this all classes are subclasses of owl:Thing.

Let us create the classes “Product” and “Brand”:

1. Navigate to the Entities tab with the Class hierarchy view selected. Make sure owl:Thing is selected.
2. Press the Add Subclass icon. This button creates a new subclass of the selected class. In this case we want to create a subclass of owl:Thing.

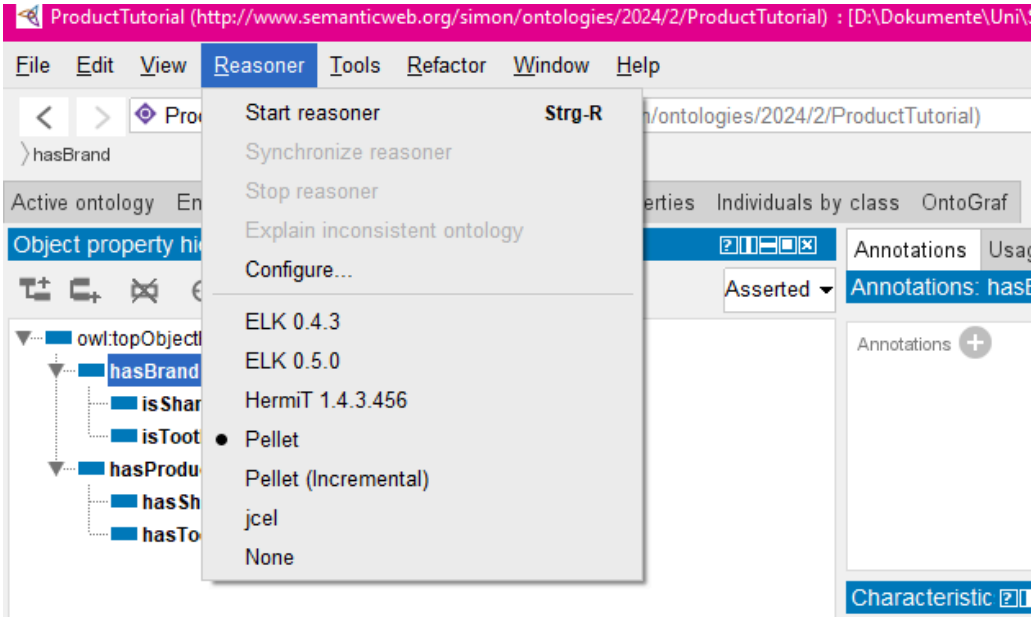
3. This should bring up a dialog titled “Create a new class” with a field for the name of the new class. Type in “Product” and then select OK.
4. Repeat these steps to create another subclass of owl:Thing: “Brand” or a sibling class to “Product”



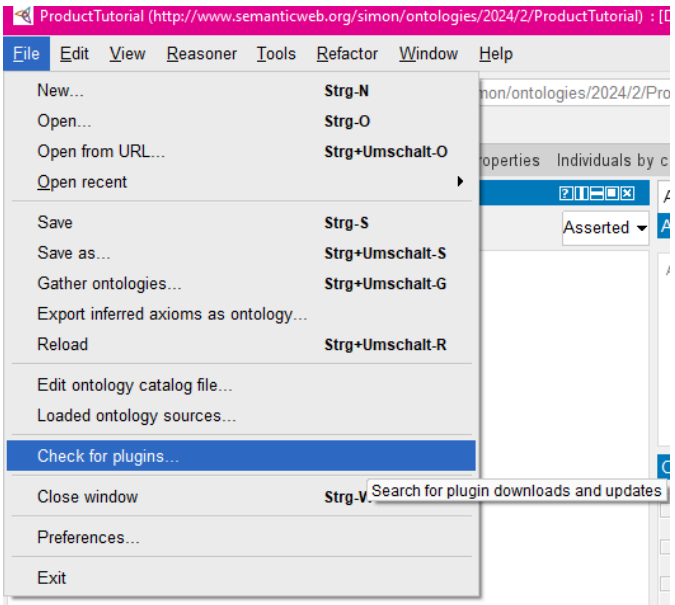
Verification: Install and Run the Pellet Reasoner

We can use a Reasoner to verify that our created classes have no inconsistencies. When just creating classes and subclasses in a new ontology, there is little chance of an inconsistency. However, it is a good idea to run the reasoner often. When there is an inconsistency, the sooner it is discovered the easier it is to fix. Let's install a reasoner and run it:

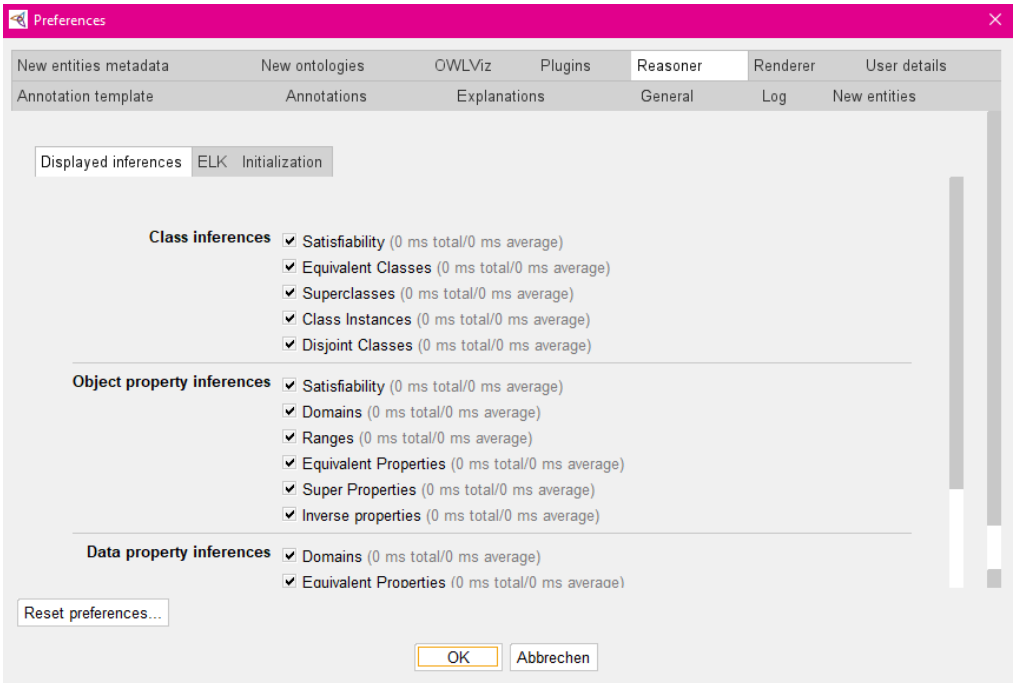
1. Check to see if the "Pellet" reasoner is installed. Click on the *"Reasoner"* menu. At the bottom of the menu there will be a list of the installed reasoners such as *Hermit* and possibly *Pellet*. If Pellet is visible in that menu then select it and skip to step 3.



2. If Pellet is not visible then go to File>Check for plugins and search for Pellet from the list of available plugins and then select Install. This will install Pellet and you should get a message that says it will take effect the next time you start Protégé. Do a File>Save to save your work, then quit Protégé and restart it. Go to File>Open recent. You should see your saved "Product Tutorial" in the list of recent ontologies. Select it to load it. Now you should see Pellet under the Reasoner menu and be able to select it.



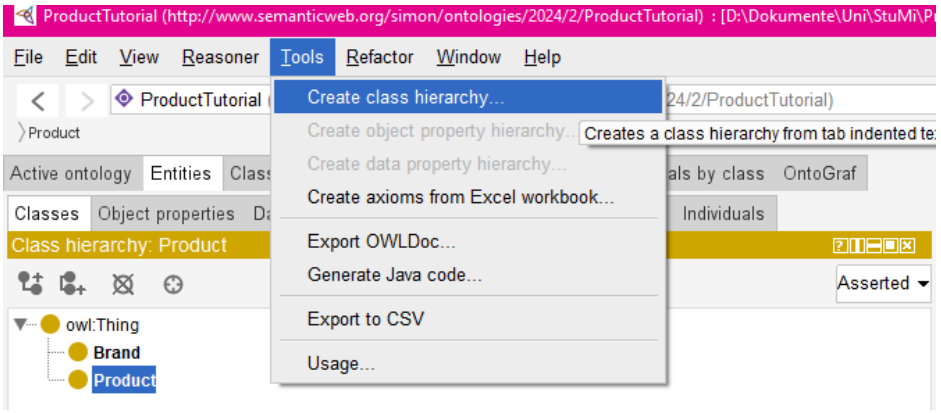
3. With Pellet selected in the Reasoner menu, execute the command Reasoner>Start reasoner. The reasoner should run very quickly since the ontology is so simple. You will notice that the little text message in the lower right corner of the Protégé window has changed to now say Reasoner active. The next time you make a change to the ontology that text will change to say: Reasoner state out of sync with active ontology. With small ontologies the reasoner runs very quickly, and it is a good idea to get into the habit of running it often.
4. It is possible that one or more of your classes will still be highlighted in red after you run the reasoner. If that happens do: Window>Refresh user interface and any red highlights should go away. Whenever your user interface seems to show something you don't expect the first thing to do is to try this command.
5. One last thing you might want to do is to configure the reasoner. By default, the reasoner does not perform all possible inferences because some inferences can take a long time for large and complex ontologies. In this tutorial we will always be dealing with small and simple ontologies so we want to see everything the reasoner can do. Go to: Reasoner>Configure. This will bring up a dialog with several checkboxes of inferences that the reasoner can perform. If they aren't all checked, for testing your first ontology you can check them all. You may receive a warning that some inferences can take a lot of time.



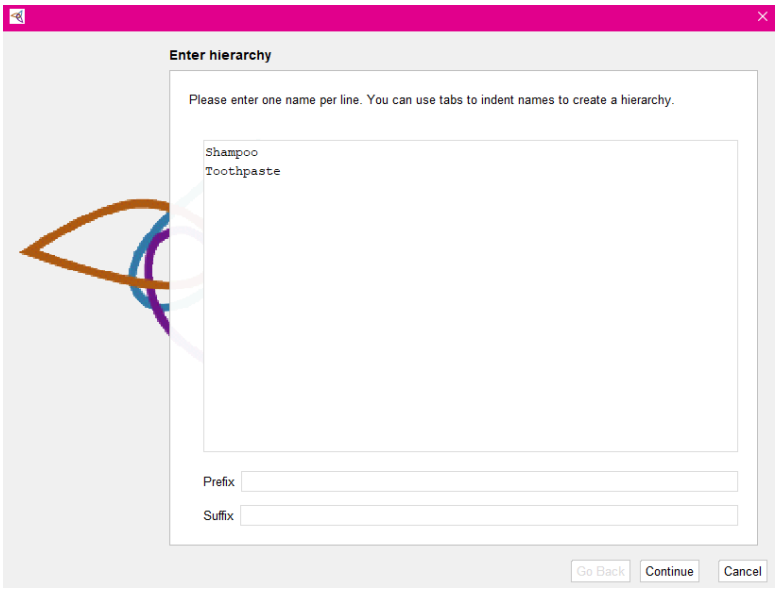
3. Create subclasses of Product

To create subclasses of the Product class, you can repeat the step in 2.. You can also use Tools>Create class hierarchy to create multiple subclasses at once.

1. Select the class *Product* in the class hierarchy.
2. With *Product* selected use the Tools>Create class hierarchy menu option



3. This should bring up a window that enables you to create a nested group of classes all at once. You should see a window labeled Enter hierarchy where you can enter one name on each line. You can also use the tab key to indicate that a class is a subclass of the class above it.
   This would look somewhat like this
   bodycare
   	shampoo
   babyproduct
   	diapers
   For now we just want to enter two subclasses of *Product*: e.g. “Shampoo” and “Toothpaste”.



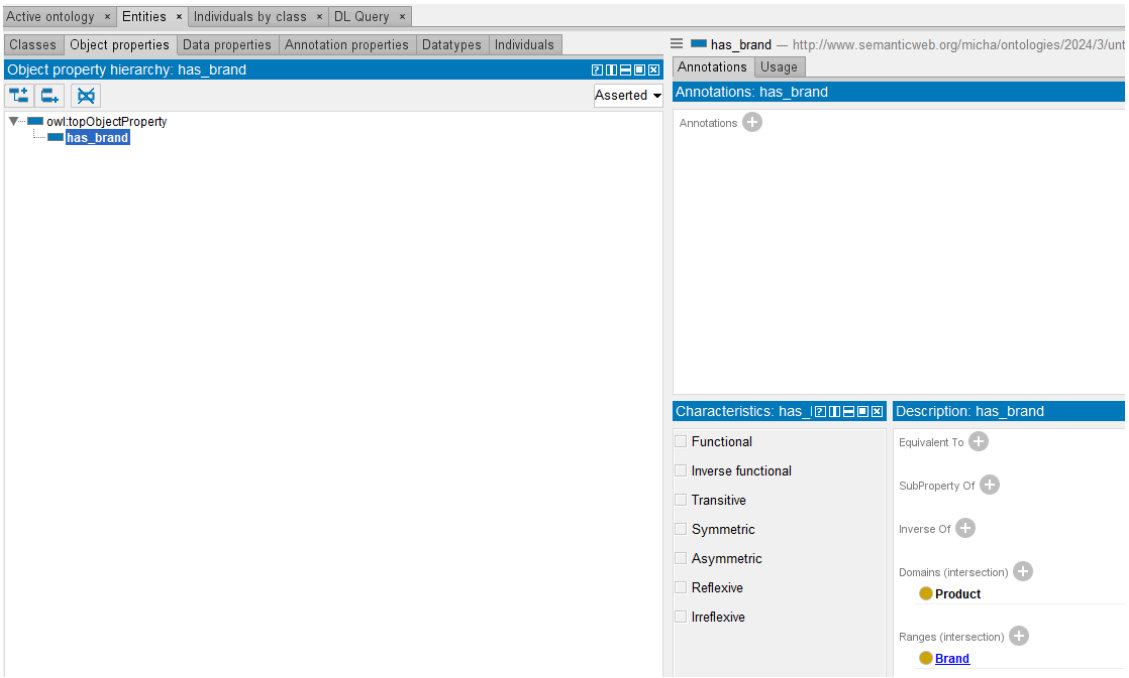
4. Select “Continue”. This will take you to a window that asks if you want to make sibling classes disjoint. Although toothpaste and shampoo products are expected to be disjoint (a product cannot be a toothpaste and a shampoo), it is recommended to not per default set all classes as disjoint classes. This can lead to problems in reasoning and querying at a later point and when using the ontology for different applications.

4. Create properties

OWL Properties represent relationships. There are three types of properties, Object properties, Data properties and Annotation properties. Object properties are relationships between two individuals. Data properties are relations between an individual and a data type such as xsd:string or xsd:dateTime. Annotation properties are descriptions of entities like a comment or a label.
We want to link the product and brand classes by stating that a product has a brand.

1. Select the *“Object properties” tab*
2. Make sure owl:topObjectProperty is selected. Click on the nested box icon at the left to create a new sub-property of “owl:topObjectProperty”. When prompted for the name of the new property type in “has_brand”
3. Just as you can use a window to create multiple classes you can also use one to create multiple properties. This can be done at *Tools>Create object property hierarchy* but is not needed for now.
4. For properties, it makes sense to define a domain and range. This helps in making the ontology consistent. A domain defines what classes the property is being assigned to, in this case the class *Product* since products have brands. Thus, the range in this case is *Brand*.

Your object property should look like this now:



Creating inverse properties

Each object property may have a corresponding inverse property. If some property links class a to class b then its inverse property will link class b to class a. The inverse property of *has_brand* would be *is_brand_of.*

1. For doing this, click on the Add icon (+) next to *Inverse Of* in the Description view for *has_brand.* You will be presented with a window that shows a nested view of all the current properties. Here, you can add your inverse property.

5. Linking Entities through relations

We now have created classes and relations for our product ontology. These have to be linked to create semantics that can be queried. One major thing to keep in mind here is that we can use two ways to create such links.

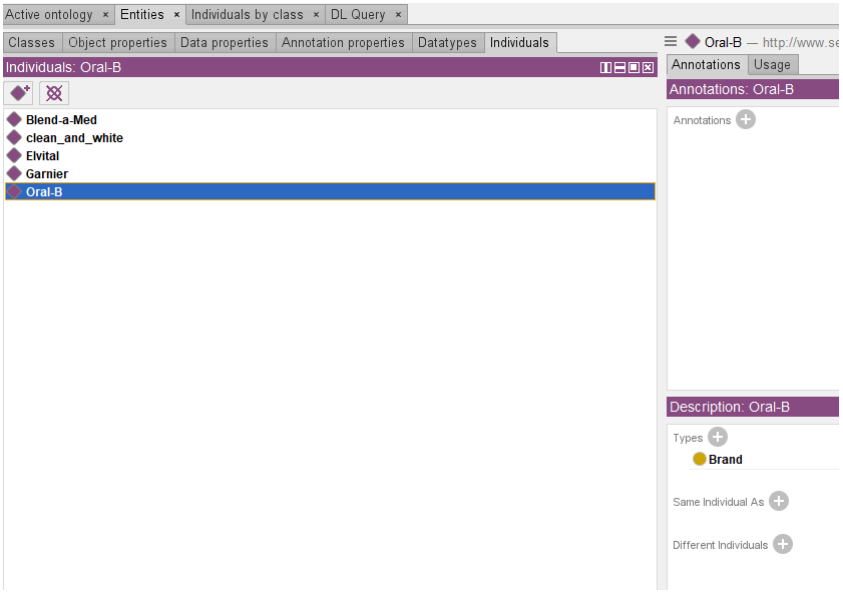
1. We can link individuals. Many take this approach. Here, one would create a shampoo individual, which then is linked to brand information. This works well for querying. However, it is crucial to exactly define and keep in mind what an individual is. If I have a “clean and white” toothpaste from blend-a-med that I want to add to the ontology, many would argue that this is an individual. From an agent perspective however, there possibly are many “clean and white” toothpastes in a house or store. The individual then is only the one I have in my hands.
2. We link classes. This usually is the way we model relations from an agent perspective. Think about the previous example. If we model “clean and white” as a subclass of toothpaste, the relation to the blend-a-med brand holds for all individuals of that class and therefore should be modeled as a class relation. Individuals of that class then are products with even more special characteristics, like different best before dates or storage positions.

5a. Linking Individuals through relations

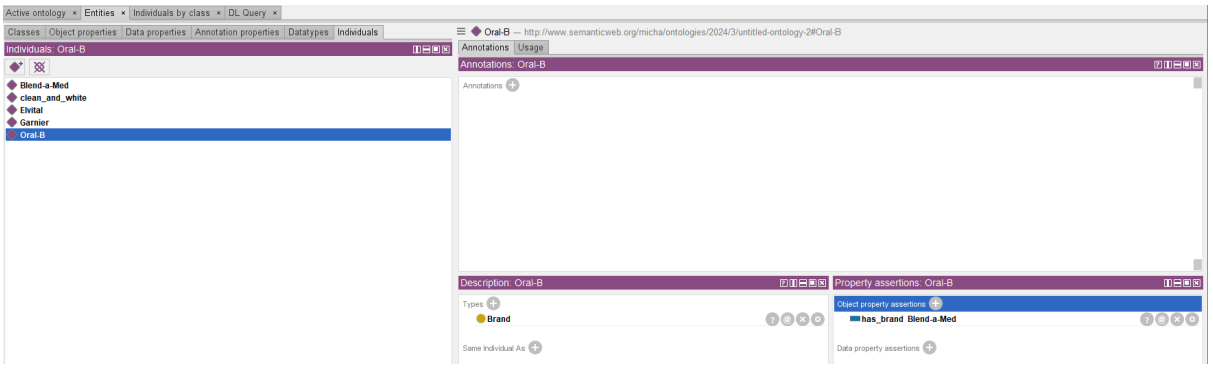
You can add an individual/instance for a class by going to the *individuals tab* or through the + button in the description of the entities. This opens a window where you can add an individual. Let’s add a “clean and white” individual here. In the individuals tab, you can now see your individual.

For this example, we also want to add brand individuals. Add something like “Blend-a-Med”, “Oral-B”, “Elvital” and “Garnier” as brand individuals. You can do this either on the class tab by pressing the + on instances in the description view or in the individuals tab by adding an individual and then assigning it a type by pressing the + in the description window here.

The individuals should look like this now:

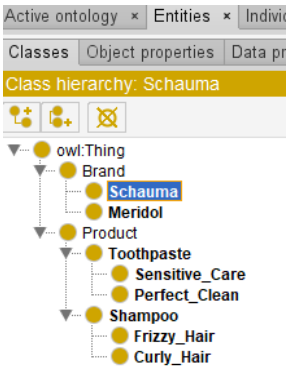


In the individual tab we can now add relations for the “clean and white” individual in the descriptions window. When pressing the + for object property annotations, a new window to add the relation and individual that shall be linked opens up.

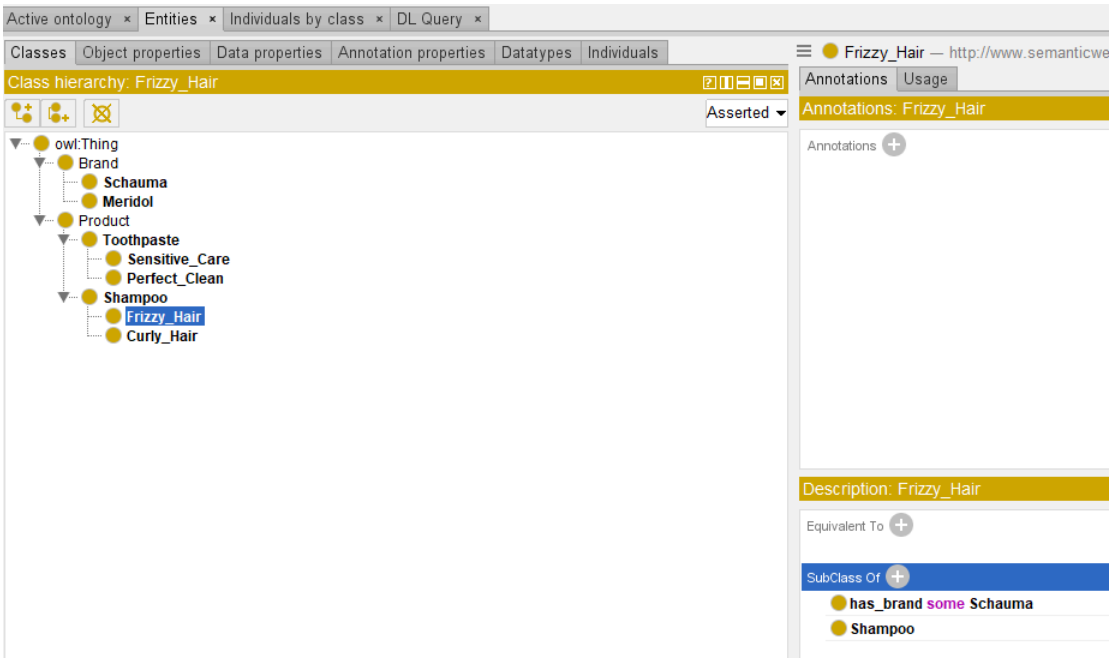


5b. Linking classes through relations

Now let us look at how to link classes for an agent application. For this, we first have to add more subclasses like we previously added individuals. Let us add a couple subclasses, such as “Schauma” and “Meridol” as brands, “Sensitive_Care” and “Perfect_Clean” as toothpastes and “Frizzy_Hair” and “Curly_hair” as shampoos.



Class relations are created in the entities tab. Here, you define a class to be the subclass of its relation. If you press the + button in the subclass of the description view, a window will pop up that lets you set different relations, like adding other parent classes or complex restrictions. Here, let us add the relation to the “Frizzy_Hair” class that it can have Schauma as brand. This is done by adding a some restriction:



This is it!

In the same manner, you can create all your classes and relations. How everything can be queried will be discussed at a later point.