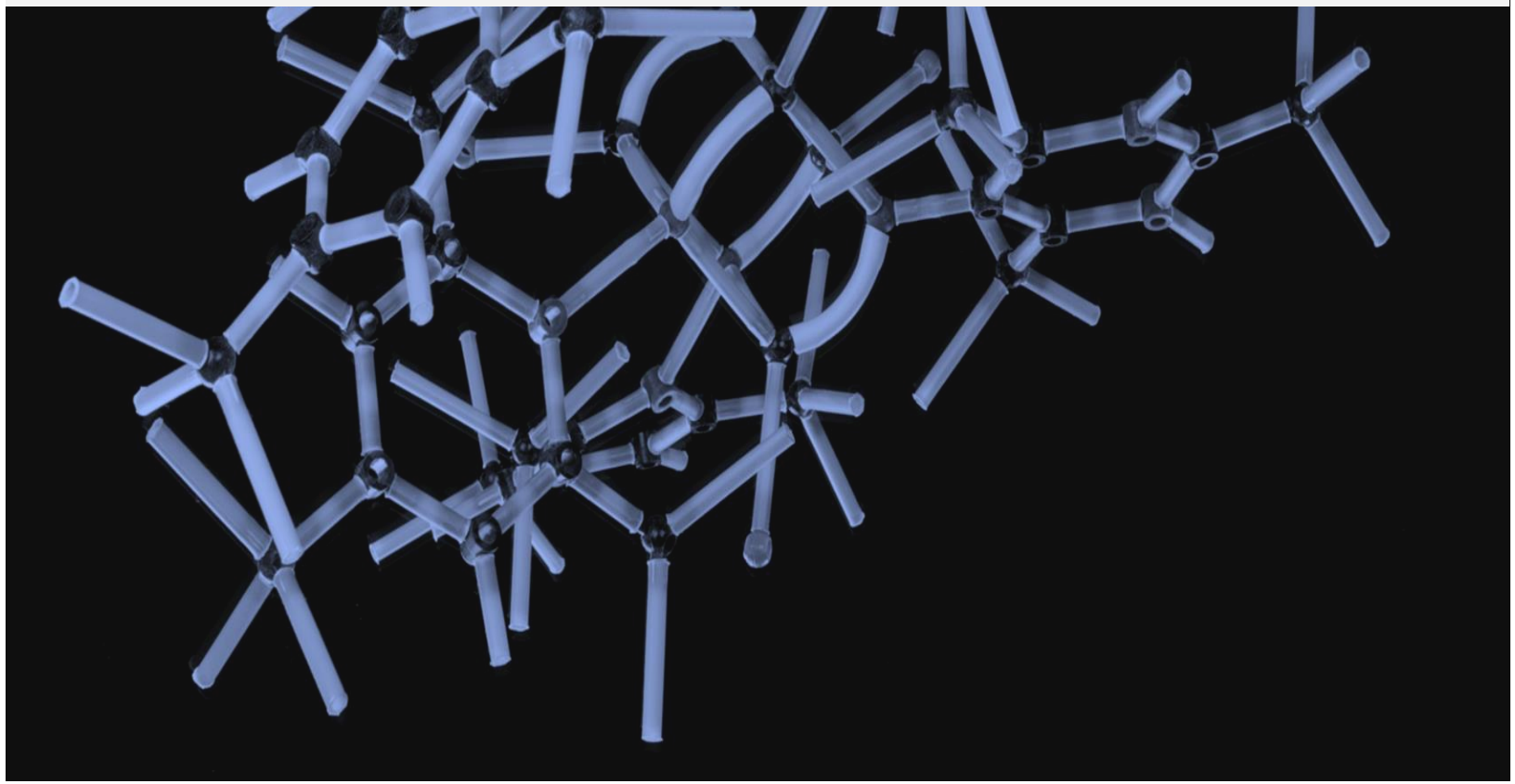


# CEDS Ontology Guide

Version 11.0.0.0

August 2023



## Introduction

The CEDS Ontology is being released as a draft for community review. The impetus for the creation of the Ontology is the result of multiple State Education Agency use cases that require the use of CEDS in a machine-readable language, such as JSON, JSON-LD, and XML. A workgroup was formed in May of 2022 to begin discussions related to the creation of JSON or XML. During that workgroup the decision was made to start the project by creating an ontology for CEDS which is the most granular description available. From that ontology, CEDS can then be expressed as JSON, JSON-LD, XML, and several other recognized standards.

The purpose of this document is to help explain some key concepts related to the Ontology during its initial review period. There is an expectation that many of the concepts, naming conventions, and diagrams in this document will change based off feedback from the CEDS community. As such, implementation of the Ontology during this draft review period is not recommended except for purposes of testing and providing feedback to CEDS.

The CEDS Ontology is comprised of six types of entities:

1. **Classes** - A Class represents a category, group, or set of individuals (also called instances) that share common characteristics, attributes, or behaviors. It serves as a conceptual grouping that defines the common features that its instances share. Classes are fundamental building blocks for creating ontologies, allowing you to organize and structure information in a logical and hierarchical manner within the semantic web context.
2. **Object Properties** - An Object Property represents a binary relation that exists between individuals (also known as instances) of different classes. Object properties are used to describe connections or links between instances, indicating some form of relationship, association, or connection between them. These properties enable the modeling of more complex relationships and interactions within ontologies, contributing to the rich representation of knowledge and data in the semantic web.
3. **Data Properties** - A Data Property represents a binary relation that associates individuals (instances) with literal values or data values. Unlike object properties, which link individuals to other individuals, data properties connect individuals to data values such as strings, numbers, dates, and other literals. Data properties are used to attribute specific data values to instances, allowing for the representation of attributes, characteristics, or data-related information within ontologies on the semantic web.
4. **Annotation Properties** - An Annotation Property is used to associate metadata or additional information with various elements within an ontology. Unlike object or data properties that relate individuals and data values, annotation properties are used to attach descriptive notes, comments, or labels to classes, individuals, properties, and other components of the ontology. These annotations do not affect the logical reasoning of the ontology but provide contextual information that aids in understanding, documentation, and communication of the ontology's content and structure.
5. **Datatypes** - A Datatype refers to a specific data type or format that specifies the kind of values that a data property can have. Datatypes define the permissible range of literal values that can be associated with individuals through data properties. These values can include strings, numbers, dates, Boolean values, and other well-defined data formats. Datatypes are crucial for

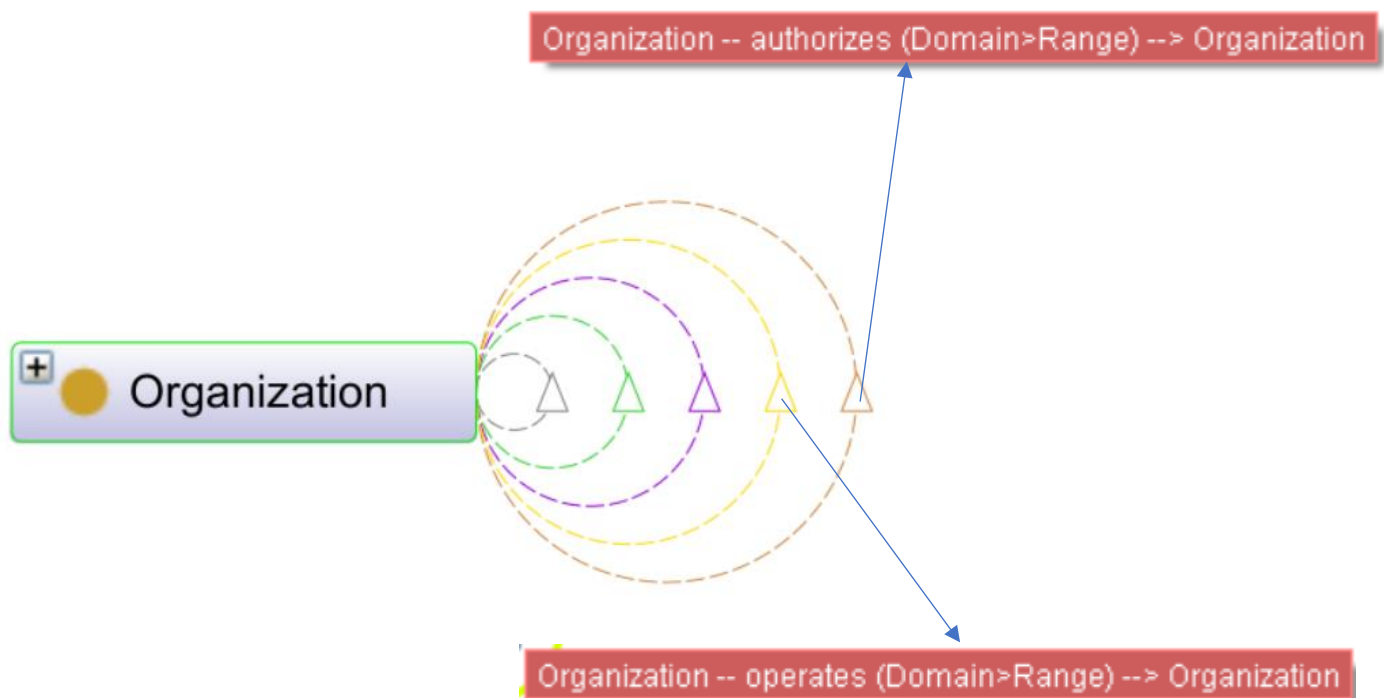
accurately representing and specifying the type of data that can be used in conjunction with data properties in an ontology.

6. **Individuals** - An Individual refers to a specific instance or entity that belongs to a particular class within an ontology. Individuals are concrete members of a class, representing distinct entities with their own identity and properties. They can be thought of as the actual instances or objects that the ontology describes, and they are used to represent real-world entities or concepts within the domain being modeled.

## Examples

### *Example 1: Organization Class with Object Property*

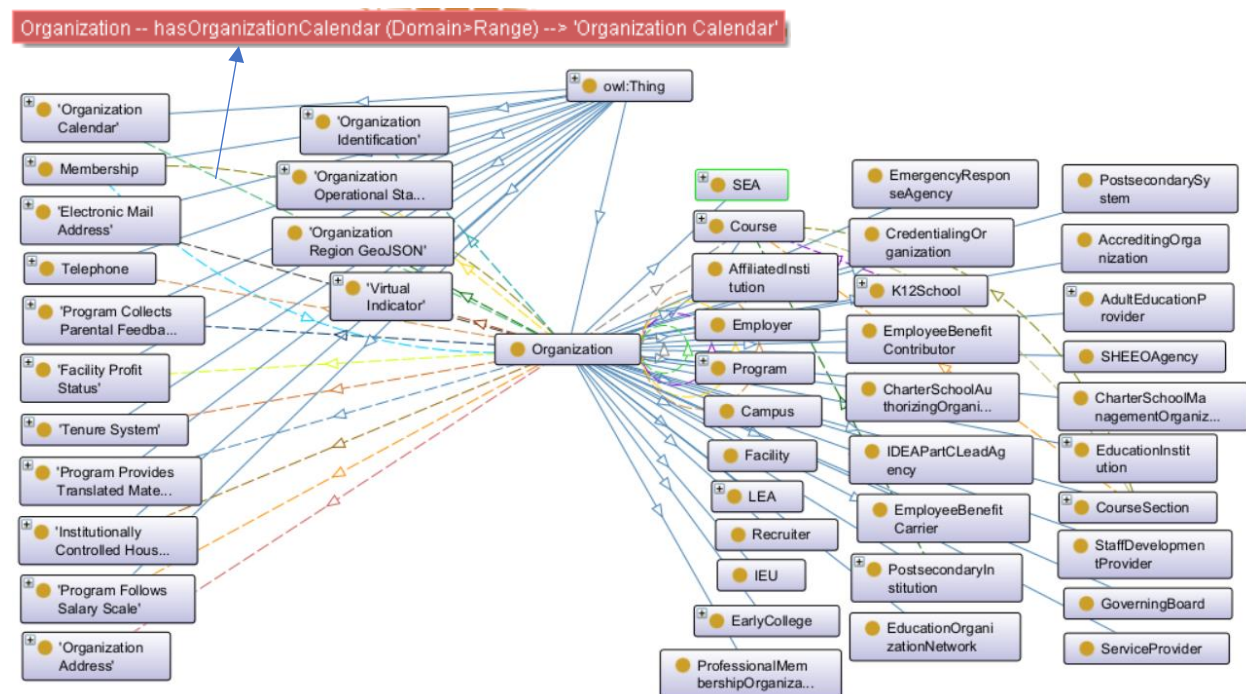
In the example below, we have a class called Organization and we have multiple circular references that relate Organization to Organization through an Object Property. The first one, “authorizes” would indicate that the first Organization authorizes the second related Organization. In a similar fashion, the second object property, “operations” indicates the first Organization operates the second related Organization.



### Example 2: Organization Class Expanded

In this example, we can see Organization expanded. On the left are classes that relate to Organization through Object Properties. On the top left, Organization Calendar relates to Organization through an Object Property called “hasOrganizationCalendar”. From a data perspective, any Organization can have a calendar. The reasoning behind creating a separate class for Organization Calendar rather than placing calendar information such as a Session Begin Date directly off Organization is that Organizations can have multiple calendars.

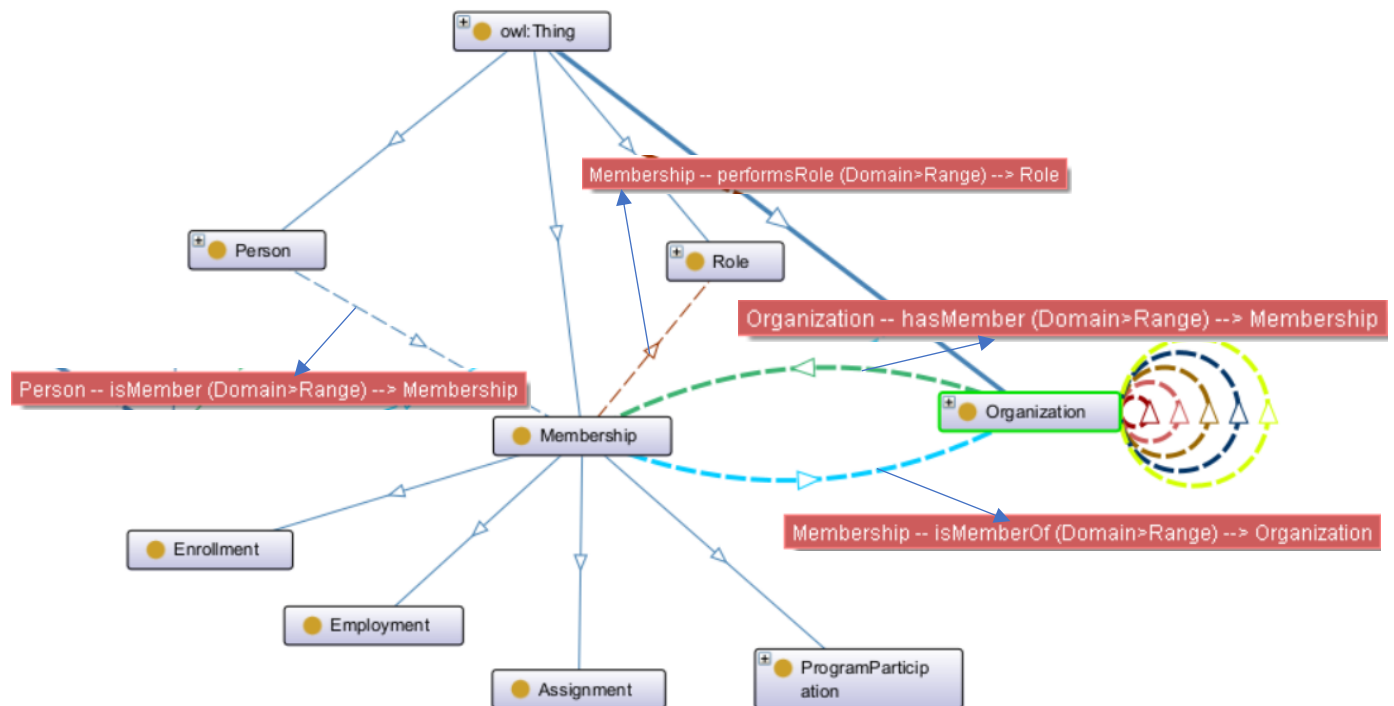
On the right side, we see sub-classes of Organization. All of these are forms of an Organization. From a data perspective, an LEA is an Organization, a Postsecondary Institution is an Organization. As future models of the Ontology are released, we can indicate elements that are applicable to all organizations, and we can also indicate elements that would only apply to specific organization making the model scalable. The element “Local Education Agency Type”, as an example would not be applicable to a Postsecondary Institution but would be applicable to the LEA whereas the element “Operational Status Effective Date” is applicable to many types of organizations and would thus be applied to the core Organization class. All sub-classes inherit properties from the superclass.



### Example 3: Organization, Membership, Person, and Role

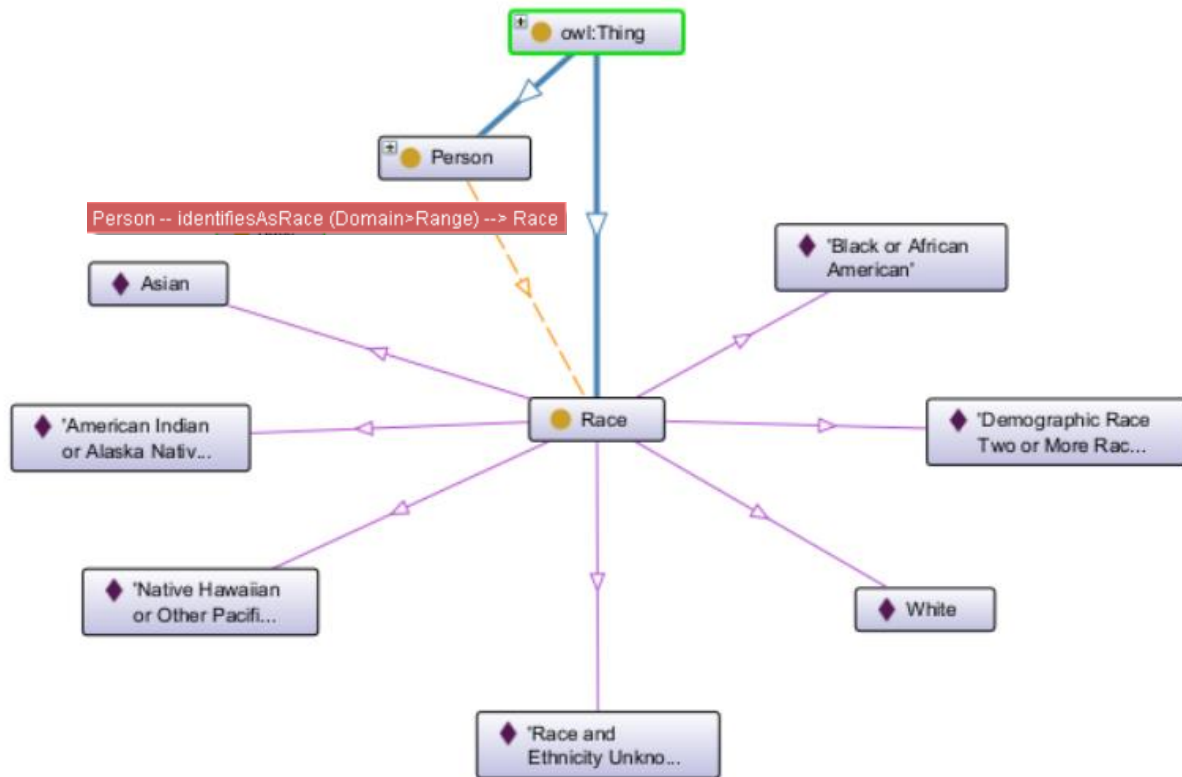
In this example, we see the primary CEDS model. A person (Class) is related to Membership (Class) through the Object Property “isMember.” Membership (Class) is related to Organization (Class) through two object properties: “isMemberOf” and “hasMember.” The Object Property “isMemberOf” explains the membership is to the Organization. The Object Property “hasMember” explains that the Organization can receive membership. Role (Class) is related to Membership (Class) through the Object Property “performsRole” indicating the role the Person plays in the Organization through that specific membership. It is possible for a person to play multiple roles in an organization but based on this model that is only done through a different membership.

We can also see in this model that Membership has four different types of sub-classes, all of which are memberships and inherit the properties of Membership: Enrollment, Employment, Assignment, ProgramParticipation.



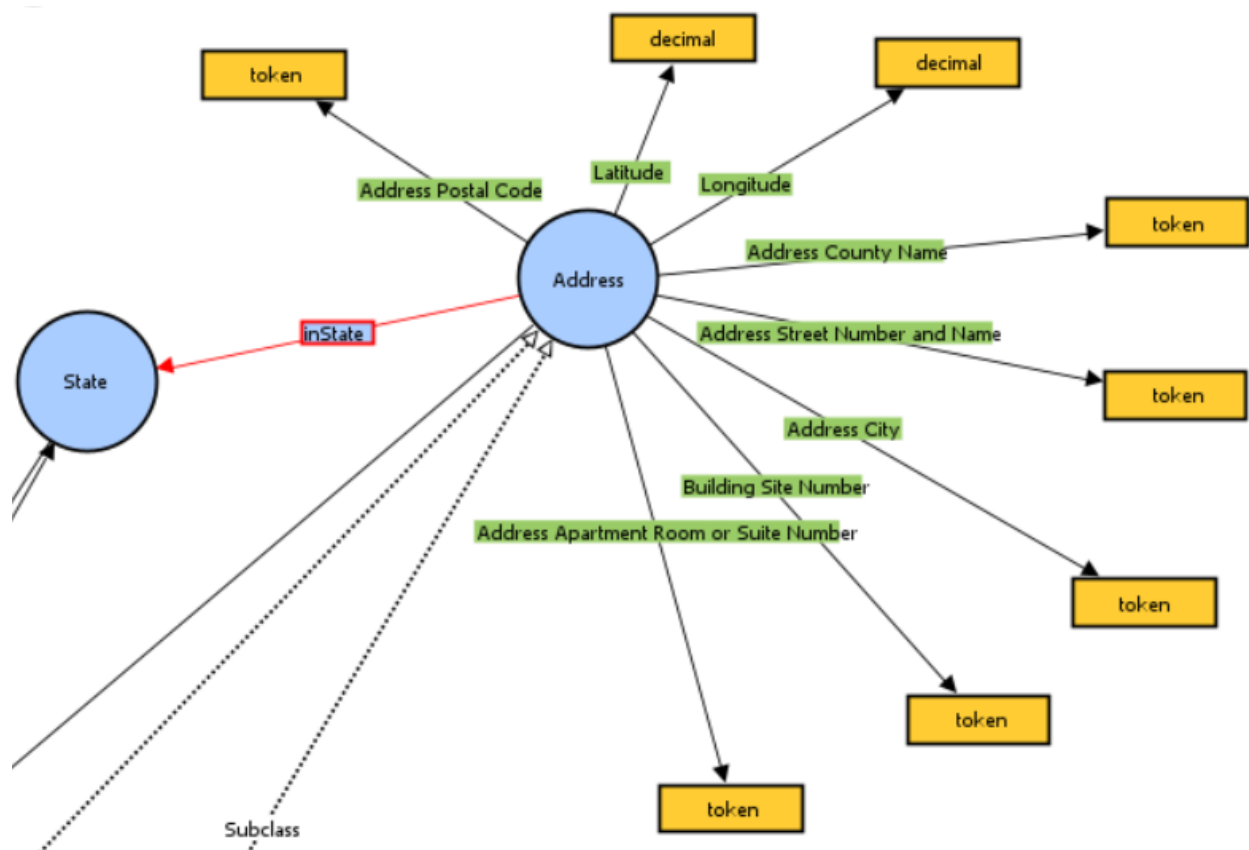
#### Example 4: Classes and Individuals

In this example, we see that a Person (Class) is related to Race (Class) through the Object Property “identifiesAsRace”. We also see the actual CEDS option sets for the Race element associated to the Race class. These are associated with Race through the Individual entity. All elements in CEDS that have an option set are created as classes. They are either created through a class/sub-class relationship or through a Class/Individual relationship.



### Example 5: Data Properties and Datatypes

In this example taken from a different graphing tool, we can see Address (Class) is surrounded by the following Data Properties: Address Postal Code, Latitude, Longitude, Address County Name, Address Street Number and Name, Address City, Building Site Number, and Address Apartment Room or Suite Number. Data Properties resolve to a literal. Another way of saying it is when an element in CEDS a text field or a number or some other type of data entry that is not an option set, it will be a Data Property. Attached to each Data Property is a Datatype that explains how the data is to be expressed, whether decimal, token, or one of many other datatypes.





### Example 6: Annotation Properties

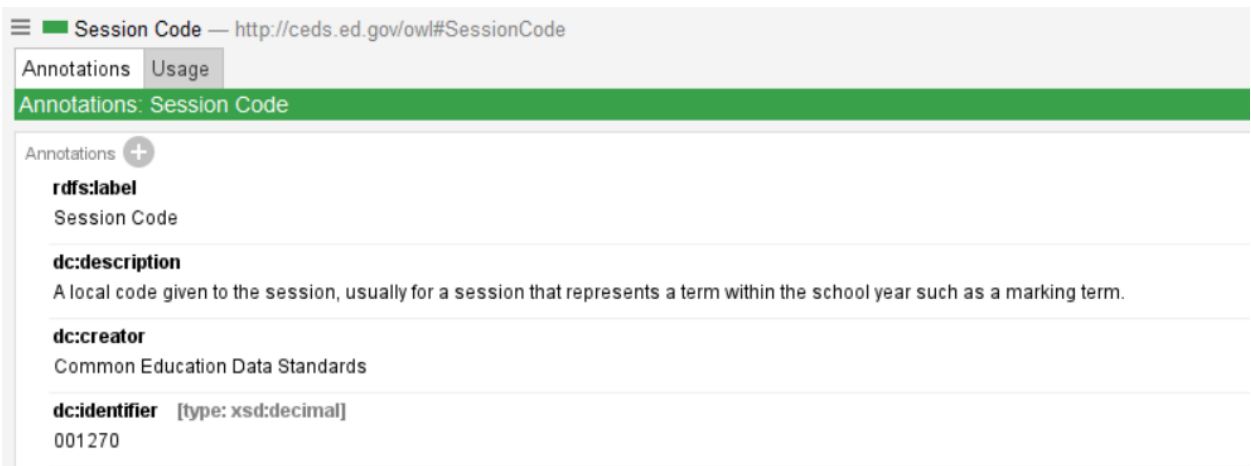
In this example, we have two images. The first shows the input screen from the open source tool called Protégé which the CEDS team has been using to build the Ontology. When complete, all Classes, Object Properties, Data Properties, and Individuals should have at least four annotations: `rdfs:label`, `dc:description`, `dc:creator`, and `dc:identifier`.

The prefixes `rdfs` and `dc` indicate the standard that defines this annotation. RDFS stands for Resource Description Framework Schema and DC stands for Dublin Core.

CEDS' use of the annotation properties is as follows:

1. `Rdfs:label` – To list the CEDS element name or CEDS Option Description
2. `Dc:description` – To list the CEDS element definition or the CEDS Option Definition
3. `Dc:creator` – Presently only lists “Common Education Data Standard” but in the future as the ontology progresses and additional standards are able to be merged, it will list the organization responsible for that entity.
4. `Dc:identifier` – To list the CEDS Global ID or the Global ID plus an “\_ (option set code)”

In the second image, we see an example of how these annotations are expressed in machine-readable RDF.



```
<!-- http://ceds.ed.gov/owl#SessionCode -->
<owl:DatatypeProperty rdf:about="http://ceds.ed.gov/owl#SessionCode">
  <rdfs:domain rdf:resource="http://ceds.ed.gov/owl#OrganizationCalendarSession"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#token"/>
  <dc:creator>Common Education Data Standards</dc:creator>
  <dc:description>A local code given to the session, usually for a session that represents a term within the school year such as a marking term.</dc:description>
  <dc:identifier rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">001270</dc:identifier>
  <rdfs:label>Session Code</rdfs:label>
</owl:DatatypeProperty>
```