Core Task Workflows and Scenarios

The tasks described below assume that the relevant domain content has already been created in the ontology according to the workflows and conventions outlined in **Annotation Properties and Individuals for driving the Application**. Here we describe the addition of axioms that facilitate the interaction of the eagle-i SWEET application with this content, in order to dynamically populate the user interface and support data collection.

In the list below, the ontology task is the primary title of each item in **bold**, and is followed in parentheses with the **application feature** or **function** that this action supports.

1) Assigning Application-Specific Labels and Definition (Drives tool tips in SWEET)

Create an annotation on the class or property of interest using the eagle-i preferred definition and eagle-i preferred label annotation properties, and enter desired text.

- These axioms should live in the application file that is paired with the domain layer file where the class or property of interest lives (e.g. ero-core-app.owl for ero.owl terms not from extended taxonomies, uberon-import-app.owl for uberon classes)
- Preferred labels should be capitalized.
- Preferred definitions are applied primarily on properties, and they are used to drive tool tips in the application. But preferred definitions can also be given to classes, and will show up in the eagle-i Ontology Browser web application (https://search.eagle-i.net/model/).

2) Flagging a Class as a Resource Type (Enables instances to be created in the SWEET)

All resources for which instances are created need an *InClassGroup* annotation axiom on the class itself (or an asserted or inferred ancestor) with the value `Class Instance Create` (`http://eagle-i.org/ont/app/1.0/ClassGroup_InstanceCreate`). These annotation axioms should live in the app layer file that is paired with the class being annotated.

> **Primary Resource Types** need a second *InClassGroup* annotation axiom with the value `Class Primary Resource Type` (`http://eagle-i.org/ont/app/1.0/ClassGroup_PrimaryResourceType`).
> Examples include 'reagent', 'software', 'instrument'.
>
> **Embedded Resource Types** need a second *InClassGroup* annotation axiom with the value `Class Embedded Resource Type` (`http://eagle-i.org/ont/app/1.0/ClassGroup_EmbeddedResourceType`)
> Examples include 'construct insert', 'phenotype annotation'.
>
> **Stubbed Resource Types** need no additional annotation axiom. Classes with only a `Class Instance Create` annotation will be treated as stubbed resources
> Examples include 'genetic alteration', 'human subject'.

> Note that these *InClassGroup* annotations are inherited by descendent classes, so they are not needed on a given class if an asserted or inferred ancestor has one. For example, the ero class 'reagent' is directly annotated with *InClassGroup* Class Instance Create and *InClassGroup* Class Primary Resource Type annotations in ero-app.owl; its inferred child 'cell line' inherits this annotation axiom and thus requires no *InClassGroup* annotations to inform the application that 'cell line' instances can be created and are primary resource types.

3) Creating a New Application-Specific Property Type (Supports data collection in the SWEET UI when the property is linked to a resource - see #4 below)
Creation of new properties is covered in the **eagle-i Resource Ontology** section. We extend these guidelines here to cover cases where the desired property is application specific and not suitable for broad community re-use, i.e. is overly specific in its meaning or domain/range constraints in order to support application needs. Unfortunately, there is not a clear rule to decide whether this is the case, and we will have to rely on the judgement of the editor and their knowledge of the ontology landscape.

In such cases, the property should live in the application layer (ero-app.owl). Additionally, a practice was partially implemented whereby the IRIs of such properties were created in a different namespace (http://eagle-i.org/ont/app/1.0/), instead of the standard http://purl.obolibrary.org/obo/ namespace. In practice, this was a good idea; however it was never consistently implemented, and only a handful of properties use this (roughly 20, e.g. http://eagle-i.org/ont/app/1.0/has_related_study_design). If it is decided to pursue this strategy, refactoring of the existing application-specific eagle-i property IRIs to use this new namespace needs to occur.

4) Linking a Resource Class to its Properties (Adds a metadata field based on that property to the form for instances of that resource class in the SWEET UI)

The fields that populate a form for collecting information about a given resource instance are determined by the domain of a given property. To get a property to appear as a metadata field for a given resource of interest in the SWEET, simply include the resource class (or an ancestor of the class) as in the domain declaration of the property.

Domain declarations are made as logical axioms that can live in the domain layer (in cases where it makes sense generally speaking for the domain of the property to be defined as such) or in the application layer (when this is not the case). As explained above, an alternative way to define the domain (or range) of a property is to create in the application layer an eagle-i domain constraint annotation that holds the IRI of the domain class as a value. Here, one annotation axiom should be made per domain to be declared, but multiple annotations can be made if the domain is a union of many classes (e.g. funded_by). This second method is useful when dealing with a community property and wish to avoid changing its definition by asserting a restricted domain or range as a logical axiom. In these cases, we want to keep these restrictions in the app layer as an annotation.

5) Defining a Property's Values (Reference Taxonomies and Inter-Resource Links) (Creates the pick-list that constrains value entry for a given controlled metadata field)

As noted in the **Viewing and Editing the Ontology in Protege** instructions, owl:DatatypeProperties are used when linking a resource to a free-text description, and owl:ObjectProperties are used when linking between entity IRIs. The types of entities that are allowed as values for a given owl:ObjectProperty are defined by the range of the property. The identity of the classes declared as the range of a property drives the pick-lists that appear in the UI when entering controlled metadata about a resource.

A) Defining Referenced Taxonomy Pick-Lists (Pointing a Property to a Referenced Taxonomy): A referenced taxonomy is a hierarchy of controlled terms used as values of properties to describe a given resource. Instances of these classes are not created in the data - rather, its IRI is used as the object of an rdf triple describing the resource. When a pick-list is needed to constrain the value set for a metadata field in the SWEET, the root class of the desired hierarchy can be added to a logical range declaration axiom. This axiom can live in the domain layer (in cases where it makes sense generally speaking for the range of the property to be defined as such) or in the application layer (when this is not the case). As we saw for defining object property domains above, an alternative way to define the range of a property is to create in the application layer an eagle-i range constraint annotation that holds the IRI of the range class as a value. Here, one annotation axiom should be made per range to be declared, but multiple annotations can be made if the domain is a union of many classes.

After declaring a given class as the range of a property, two specific annotations are required on the class to make it the root of a referenced taxonomy:

- An *InClassGroup* annotation with the value `Class Referenced Taxonomy` is needed.
  - If the taxonomy root class lives in the core ero.owl file, this annotation should live only in the eagle-i-core-app.owl file (e.g. 'technique', 'data format specification').
  - If the taxonomy root class lives in an extended owl file (e.g. 'material anatomical entity' in uberon-import.owl), the annotation needs to live in the appropriate application layer owl file (e.g. uberon-import-app.owl), and then duplicated in the eagle-i-core-app.owl file.
- An eagle-i referenced taxonomy IRI annotation holding the IRI of the application ontology for the extended ontology where the taxonomy content lives is also needed on the root of the referenced taxonomy.
  - For a class in the core ero.owl file, the value of this annotation will be "http://eagle-i.org/ont/app/1.0/eagle-i-core-app.owl"
  - For a class in an extended owl file, the value will the IRI of the appropriate application layer file (e.g. "http://eagle-i.org/ont/app/1.0/uberon-import-app.owl" for a referenced taxonomy rooted in uberon-import.owl). This annotation lives in the eagle-i-core-app.owl file. The IRIs of all app layer ontology files that might be values of the eagle-i referenced taxonomy IRI property can be found in Table 3 above.

For an example of how this works for a referenced taxonomy rooted in an extended owl file, see the 'material anatomical entity' class. Here, the *InClassGroup* annotation lives in the uberon-import-app.owl file AND the eagle-i-core-app.owl file. And the eagle-i referenced taxonomy IRI annotation lives in the eagle-i-core-app.owl file, and holds the value "http://eagle-i.org/ont/app/1.0/uberon-import-app.owl". The duplicated *InClassGroup* annotation is needed so that the root of the referenced taxonomy is flagged in a file that is loaded into memory, and then the eagle-i referenced taxonomy IRI annotation tells the application logic where to go to retrieve the content of this referenced taxonomy. Compare this to a referenced taxonomy rooted in the core ero.owl file such as 'technique' or 'data format specification' - where the duplication is not required.

An important variation on this workflow is pointing a property to a sub-hierarchy of an existing referenced taxonomy: When pointing a property at a descendant of a core ero.owl class that is already declared as the root of a referenced taxonomy, no additional annotations are needed if the class lives in the core ero.owl file. However, if the class lives in an extended owl file, an additional eagle-i referenced taxonomy IRI annotation holding the IRI of the relevant app layer file is required to direct the application logic to this file and load it into memory (e.g. "http://eagle-i.org/ont/app/1.0/uberon-import-app.owl" for a referenced taxonomy rooted in uberon-import.owl). An example of this can be found on the 'germ layer/neural crest' class that is a descendant of the 'material anatomical entity' class, and declared as the range of the has_potency property that hangs from 'stem cell lines'

B) Defining Resource-Instance Pick Lists / Inter-Resource Links (Pointing a Property to Another Resource):
Enabling a metadata field for a given resource to take a related resource IRI as a value is an important feature of the eagle-i application, as it creates inter-resource links in the data that support navigation and builds a true knowledge graph. This is achieved by first pointing a property from a resource (the domain) to the class of resources with which the link is desired (the range). To do this use one of the two mechanisms for defining property domains ranges as described above. The target class is a resource and therefore should already have an *InClassGroup* Class Instance Create annotation axiom, along with an additional *InClassGroup* annotation defining it as Primary or Embedded where applicable. Details on creating these annotations are provided above. With these annotations in place, the SWEET will recognize the specified metadata field as linking form a resource instance to another resource instance, and provide a picklist of existing resource instances (along with a means to create a new resource instance with which to link).

6) Extending a Referenced Taxonomy (Adds values to an existing picklist list)

Simply add the new class(es) to the file where the taxonomy lives, along with any application metadata (preferred labels or definitions). No additional annotations are needed, as the class will inherit the referenced taxonomy annotations from the hierarchy root. Upon the next application build, the new class should appear in the appropriate picklist in the SWEET UI.