ontology-development-kit Documentation

Release 0.5

ontology-development-kit

Nov 19, 2020

Contents:

This guide is intended for maintainers and editors of OBO ontologies.

The first part of the guide is how to set up a new ontology project. This requires some technical steps.

The second part of the guide is for day to day editing.

NOTE this guide is adapted from the GO editors guide. Some instructions may still be too GO-specific.

CHAPTER 1

Creating a new Ontology using the ODK

We will walk you though the steps to make a new ontology project

1.1 1. Install and Start Docker

- docker

See below for an alternative protocol where you install the software yourself rather than use Docker

1.2 2. Download the wrapper script and pull latest ODK version

- Linux/Mac: seed-via-docker.sh
- PC: seed-via-docker.bat
- First, make sure you have Docker running (you will see the Docker whale in your toolbar on a Mac)
- To make sure you have the latest version of the ODK installed, run in the command line

  `docker pull obolibrary/odkfull`

NOTE The very first time you run this it may be slow, while docker downloads necessary images. Don't worry, subsequent runs should be much faster!

1.3 3. Run the wrapper script

You can either pass in a `project.yaml` file that specifies your ontology project setup, or you can pass arguments on the command line.

Passing arguments on the command line:

```
./seed-via-docker.sh -d po -d ro -d pato -u cmungall -t "Triffid Behavior ontology"␣
↪triffo
```

Using a the predefined examples/triffo/project.yaml file:

```
./seed-via-docker.sh -C examples/triffo/project.yaml
```

You can add a -c (lowercase) just before the -C (capital c) in the command to first delete any previous attempt to generate your ontology with the ODK, and then replaces it with a completely new one.

This will create your starter files in `target/triffid-behavior-ontology`. It will also prepare an initial release and initialize a local repository (not yet pushed to your Git host site such as GitHub or GitLab).

You can customize at this stage, or (recommended) after making an initial push to your git host.

1.4 4. Push to Git hosting website

The development kit will automatically initialize a git project, add all files and commit.

You will need to create a project on you Git hosting site.

For GitHub:

1. Go to: https://github.com/new
2. The owner MUST be the org you selected with the `-u` option. The name MUST be the one you set with `-t`.
3. Do not initialize with a README (you already have one)
4. Click Create
5. See the section under "…or push an existing repository from the command line"

For GitLab:

1. Go to: https://gitlab.com/projects/new
2. The owner MUST be the org you selected with the `-u` option. The name MUST be the one you set with `-t`.
3. Do not initialize with a README (you already have one)
4. Click 'Create project'
5. See the section under "Push an existing Git repository"

Follow the instructions there. E.g.

```
cd target/triffid-behavior-ontology
git remote add origin git@github.com:cmungall/triffid-behavior-ontology.git
git push -u origin master
```

Note: you can now mv `target/triffid-behavior-ontology` to anywhere you like in your home directory. Or you can do a fresh checkout from github.

CHAPTER 2

Initial Setup for Ontology Developers

2.1 Installing Protege

1. Follow the instructions on the GO wiki page: http://wiki.geneontology.org/index.php/Protege_setup_for_GO_Eds
2. *Need to add more here about the different Views and Tabs needed for working.*

2.2 ID Ranges

1. Curators and projects are assigned specific ID ranges within the prefix for your ontology. See the README-editors.html for your ontology
2. An example: go-idranges.owl
3. **NOTE:** You should only use IDs within your range.
4. If you have only just set up this repository, modify the idranges file and add yourself or other editors.

2.3 Setting ID ranges in Protege

1. Once you have your assigned ID range, you need to configure Protege so that your ID range is recorded in the Preferences menu. Protege does not read the idranges file.
2. In the Protege menu, select Preferences.
3. In the resulting pop-up window, click on the New Entities tab and set the values as follows.
4. In the Entity IRI box:

   Start with: Specified IRI: http://purl.obolibrary.org/obo

   Followed by: `/`

   End with: `Auto-generated ID`

5. In the Entity Label section:

   Same as label renderer: IRI: http://www.w3.org/2000/01/rdf-schema#label

6. In the Auto-generated ID section:

   - Numeric
   - Prefix `GO_`
   - Suffix: *leave this blank*
   - Digit Count `7`
   - **Start:** see go-idranges.owl. Only paste the number after the `GO:` prefix. Also, note that when you paste in your GO ID range, the number will automatically be converted to a standard number, e.g. pasting 0110001 will be converted to 110,001.)
   - **End:** see go-idranges.owl
   - Remember last ID between Protege sessions: ALWAYS CHECK THIS

   (**Note:** You want the ID to be remembered to prevent clashes when working in parallel on branches.)

NOTE This documentation is incomplete, for now you may be better consulting the [http://wiki.geneontology.org/index.php/Protege_setup_for_GO_Eds](GO Editor Docs)

CHAPTER 3

Configuration

3.1 Configuring New Entities Metadata

1. In the Protege menu, select Preferences.
2. Click on: Annotate new entities with creator (user)
3. Creator property: Add http://www.geneontology.org/formats/oboInOwl#created_by
4. Creator value: Use user name
5. Check: Annotate new entities with creation date and time.
6. Date property: Add http://www.geneontology.org/formats/oboInOwl#creation_date
7. Check: ISO-8601

3.2 Configuring User details

Select 'User name', and use the supplied user name; that is, your GOC identity.

3.2.1 Identifying the user for commits

Git needs to know who is committing changes to the repository, so the first time you commit, you may see the following message:

```
  Committer: Kimberly Van Auken <vanauken@kimberlukensmbp.dhcp.lbnl.us>
    Your name and email address were configured automatically based on your username␣
↪and hostname. Please check that they are accurate.
```

You can suppress this message by setting your name and email explicitly:

1. Type `git config --global user.name "Your Name"`

2. Type `git config --global user.email you@example.com`.
3. You can then fix the identity used for this commit by typing: `git commit --amend --reset-author`.

NOTE This documentation is incomplete, for now you may be better consulting the [http://wiki.geneontology.org/index.php/Installing_and_Using_git](GO Editor Docs]

CHAPTER 4

Installing and Using git

4.1 Installing git

1. In order to locally edit the ontology and push changes back to the GitHub repository, you will need to have git installed on your machine.
2. To check if you already have git installed, or to see what version of git you have, type either of these commands in your terminal: `which git` or `git --version`.
3. To install git, follow instructions here: https://git-scm.com/

Note for MacOSX users: it is advised to install Xcode tools.

4.2 Cloning the go-ontology repository from GitHub

1. Create a directory called `repos` on your local machine using this command: `mkdir repos`.
2. Then paste this command into your terminal: `git clone https://github.com/geneontology/go-ontology.git`.

   Example result:

```
Cloning into 'go-ontology'...
remote: Counting objects: 2541, done.
remote: Compressing objects: 100% (100/100), done.
remote: Total 2541 (delta 52), reused 0 (delta 0), pack-reused 2440
Receiving objects: 100% (2541/2541), 21.19 MiB | 5.22 MiB/s, done.
Resolving deltas: 100% (1532/1532), done.
```

4.3 Editing the .profile (or .bashrc) file to indicate the branch you are working on

It can be very helpful to know what branch you are working in on your terminal window. You can set this up to display by adding the following information to your .profile file (found by typing ls -a):

```
export GO_REPO=~/repos/MY-ONTOLOGY
. $GO_REPO/src/util/git-completion.bash
parse_git_branch() {
    git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/(\1)/'
}
PS1="\w\$(parse_git_branch) $ "
export PATH=$PATH:$HOME/bin/
```

Note the last line is not relevant to git, but we do this now for later on when we want to run tools like robot.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 5

Daily Workflow

5.1 Updating the local copy of the ontology with 'git pull'

1. Navigate to the ontology directory of go-ontology: `cd repos/MY-ONTOLOGY/src/ontology`.
2. If the terminal window is not configured to display the branch name, type: `git status`. You will see:

```
On branch [master] [or the name of the branch you are on]
Your branch is up-to-date with 'origin/master'.
```

3. If you're not in the master branch, type: `git checkout master`.
4. From the master branch, type: `git pull`. This will update your master branch, and all working branches, with the files that are most current on GitHub, bringing in and merging any changes that were made since you last pulled the repository using the command `git pull`. You will see something like this:

```
~/repos/MY-ONTOLOGY(master) $ git pull
remote: Counting objects: 26, done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 26 (delta 12), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (26/26), done.
From https://github.com/geneontology/go-ontology
   580c01d..7225e89  master         -> origin/master
 * [new branch]          issue#13029 -> origin/issue#13029
Updating 580c01d..7225e89
Fast-forward
 src/ontology/go-edit.obo | 39 ++++++++++++++++++++++++---------------
 1 file changed, 24 insertions(+), 15 deletions(-)
~/repos/MY-ONTOLOGY(master) $
```

5.2 Creating a New Working Branch with 'git checkout'

1. When starting to work on a ticket, you should create a new branch of the repository to edit the ontology file.

2. **Make sure you are on the master branch before creating a new branch**. If the terminal window is not configured to display the branch name, type: `git status` to check which is the active branch. If necessary, go to master by typing `git checkout master`.
3. To create a new branch, type: `git checkout -b issue-NNNNN` in the terminal window. For **naming branches**, we recommend using the string 'issue-' followed by the issue number. For instance, for this issue in the tracker: https://github.com/geneontology/go-ontology/issues/13390, you would create this branch: `git checkout -b issue-13390`. Typing this command will automatically put you in the new branch. You will see this message in your terminal window:

```
~/repos/MY-ONTOLOGY/src/ontology(master) $ git checkout -b issue-13390
Switched to a new branch 'issue-13390'
~/repos/MY-ONTOLOGY/src/ontology(issue-13390) $
```

5.3 Continuing work on an existing Working Branch

1. If you are continuing to do work on an existing branch, in addition to updating master, go to your branch by typing `git checkout [branch name]`. Note that you can view the existing local branches by typing `git branch -l`.
2. **OPTIONAL**: To update the working branch with respect to the current version of the ontology, type `git pull origin master`. This step is optional because it is not necessary to work on the current version of the ontology; all changes will be synchronized when git merge is performed.

5.4 Loading, navigating and saving the Ontology in Protégé

1. Before launching Protégé, make sure you are in the correct branch. To check the active branch, type `git status`.
2. Click on the 'File' pulldown. Open the file: go-edit.obo. The first time, you will have to navigate to `repos/MY-ONTOLOGY/src/ontology`. Once you have worked on the file, it will show up in the menu under 'Open'/'Recent'.
3. Click on the 'Classes' tab.
4. Searching: Use the search box on the upper right to search for a term in the ontology. Wait for autocomplete to work in the pop-up window.
5. Viewing a term: Double-click on the term. This will reveal the term in the 'Class hierarchy' window after a few seconds.
6. Launching the reasoner: To see the term in the 'Class hierarchy' (inferred) window, you will need to run the 'ELK reasoner'. 'Reasoner' > select ELK 0.4.3, then click 'Start reasoner'. Close the various pop-up warnings about the ELK reasoner. You will now see the terms in the inferred hierarchy.
7. After modification of the ontology, synchronize the reasoner. Go to menu: 'Reasoner' > ' Synchronize reasoner'.
   - **NOTE**: The only changes that the reasoner will detect are those impacting the ontology structure: changes in equivalence axioms, subclasses, merges, obsoletions, new terms.
   - **TIP**: When adding new relations/axioms, 'Synchronize' the reasoner. When deleting relations/axioms, it is more reliable to 'Stop' and 'Start' the reasoner again.
8. Use File > Save to save your changes.

5.5 Committing, pushing and merging your changes to the repository

1. **Review**: Changes made to the ontology can be viewed by typing `git diff` in the terminal window. If there are changes that have already been committed, the changes in the active branch relative to master can be viewed by typing `git diff master`.

2. **Commit**: Changes can be committed by typing: `git commit -m ‘Meaningful message Fixes #ticketnumber’ go-edit.obo`.

   For example:

   ```
   git commit -m ‘hepatic stellate cell migration and contraction and regulation␣
   ↪terms. Fixes #13390’ go-edit.obo
   ```

   This will save the changes to the go-edit.obo file. The terminal window will show something like:

   ```
   ~/repos/MY-ONTOLOGY/src/ontology(issue-13390) $ git commit -m 'Added hepatic␣
   ↪stellate cell migration and contraction and regulation terms. Fixes #13390' go-
   ↪edit.obo
   [issue-13390 dec9df0] Added hepatic stellate cell migration and contraction and␣
   ↪regulation terms. Fixes #13390
   1 file changed, 79 insertions(+)
   ~/repos/MY-ONTOLOGY/src/ontology(issue-13390) $
   ```

   - **NOTE**: The word ‘fixes’ is a magic word in GitHub; when used in combination with the ticket number, it will automatically close the ticket. In the above example, when the file is merged in GitHub, it will close issue number 13390. Learn more on this GitHub Help Documentation page about ‘Closing issues via commit messages’.
   - ‘Fixes’ is case-insensitive.
   - If you don’t want to close the ticket, just refer to the ticket # without the word ‘Fixes’. The commit will be associated with the correct ticket but the ticket will remain open.
   - **NOTE**: It is also possible to type a longer message than allowed when using the ‘-m’ argument; to do this, skip the -m, and a vi window (on mac) will open in which an unlimited description may be typed.
   - **TIP**: Git needs to know who is committing changes to the repository, so the first time you commit, you may see the following message:

     ```
     Committer: Kimberly Van Auken <vanauken@kimberlukensmbp.dhcp.lbnl.us>
        Your name and email address were configured automatically based on your␣
     ↪username and hostname. Please check that they are accurate.
     ```

   - See Configuration instructions to specify your name and email address.

3. **Push**: To incorporate the changes into the remote repository, type: `git push origin mynewbranch`.

   Example:

   ```
   git push origin issue-13390
   ```

   - **TIP**: Once you have pushed your changes to the repository, they are available for everyone to see, so at this stage you can ask for feedback.

4. **Pull**

   1. In your browser, return to the GO Ontology repository on GitHub.

   2. Navigate to the tab labeled as 'Code' `geneontology/go-ontology/code`. You will see your commit listed at the top of the page in a light yellow box. If you don't see it, click on the 'Branches' link to reveal it in the list, and click on it.
   3. Click the green button 'Compare & pull request' on the right.
   4. You may now add comments and ask a colleague to review your pull request. If you want to have the ticket reviewed before closing it, you can select a reviewer for the ticket before you make the pull request by clicking on the 'Reviewers' list and entering a GitHub identifier (e.g. @superuser1). The reviewer will be notified when the pull request is submitted. Since the Pull Request is also a GitHub issue, the reviewer's comments will show up in the dialog tab of the pull request, similarly to any other issue filed on the tracker.
   5. The diff for your file is at the bottom of the page. Examine it as a sanity check.
   6. Click on the green box 'Pull request' to generate a pull request.
   7. Wait for the Travis checks to complete (this can take a few minutes). If the Travis checks failed, go back to your working copy and correct the reported errrors.

5. **Merge** If the Travis checks are succesful and **if you are done working on that branch**, merge the pull request. Confirming the merge will close the ticket if you have used the word 'fixes' in your commit comment. **NOTE**: Merge the branches only when the work is completed. If there is related work to be done as a follow up to the original request, create a new GitHub ticket and start the process from the beginning.
6. **Delete** your branch on the repository using the button on the right of the successful merge message.
7. You may also delete the working branch on your local copy. Note that this step is optional. However, if you wish to delete branches on your local machine, in your terminal window:
   1. Go back to the master branch by typing `git checkout master`.
   2. Update your local copy of the repository by typing `git pull origin master`
   3. Delete the branch by typing `git branch -d workingbranchname`. Example: `git branch -d issue-13390`

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 6

Creating a New Ontology Term

See Daily Workflow for creating branches and basic Protégé instructions.

1. To create a new term, the 'Asserted view' must be active (not the 'Inferred view').
2. In the Class hierarchy window, click on the 'Add subclass' button at the upper left of the window.
3. A pop-up window will appear asking you to enter the Name of the new term. When you enter the term name, you will see your ID automatically populate the IRI box. Once you have entered the term, click 'OK' to save the new term. You will see it appear in the class hierarchy.
4. Navigate to the OBO annotation window.
5. In the OBO annotation window add:
    1. Namespace
        1. Begin typing one of the three branches (autocomplete will guide you to the correct term): - `biological_process` - `cellular _component` - `molecular_function`
        2. For Type, select: `xsd:string`
    2. Definition
        1. Click on the + next to the Definition box
        2. Add the textual definition in the pop-up box.
        3. For Type, select: `Xsd:string`
        4. Click OK.
    3. Add Definition References
        1. Click on the circle with the 'Ref' in it next to add Definition References and in the resulting pop-up click on the + to add a new ref, making sure they are properly formatted with a database abbreviation followed by a colon, followed by the text string or ID. Examples: `GOC:bhm, PMID:27450630`.
        2. Select Type: `xsd:string`
        3. Click OK.

4. Add each definition reference separately by clicking on the + sign.

4. Add synonyms and dbxrefs following the same procedure if they are required for the term.

6. If you have created a logical definition for your term, you can delete the asserted `is_a` parent in the 'subclass of' section. Once you synchronize the reasoner, you will see the automated classification of your new term. If the inferred classification doesn't make sense, then you will need to modify the logical definition.

```
Protege tip:  If you need to create a logical definition using a GO term name␣
↪that does not begin
with an alphabetic character, e.g. GO:0004534 (5'-3' exoribonuclease activity),␣
↪navigate to the
View menu in Protege and select 'Render by entity IRI short name (Id).  This will␣
↪allow you to
enter a logical definition by entering the relations and term as IDs, e.g. RO_
↪0002215 some GO_0004534.
Note the use of the underscore instead of the colon in the ID.  You can then␣
↪return to the View
menu to switch back to Render by label (rdfs:label) to see the term names.
```

7. In some cases such as `part_of` relations based on external partonomies, it might be necessary to assert the `part_of` relationships. For example: `'heart valve development' part_of some 'heart development'`. In those cases, it is important to browse the external ontologies to be sure that nothing is missing.

8. When you have finished adding the term, you can hover over it in the class window to reveal its GO_id.

9. Save the file and ***return to your terminal window***. Then, type: `git status`. This will confirm which file has been modified.

10. To see how the branch was modified, type: `git diff`. In this case, go-edit.obo was modified. The text below is not the entire diff for this edit, but is an example. If the diff is very large, you will need to hit a space to continue to see it and then hit 'q' to get back to the prompt at the end of the diff file.

```
~/repos/MY-ONTOLOGY/src/ontology(issue-13390) $ git diff
diff --git a/src/ontology/go-edit.obo b/src/ontology/go-edit.obo
index 72ae7e9..8d47fa1 100644
--- a/src/ontology/go-edit.obo
+++ b/src/ontology/go-edit.obo
@@ -400751,6 +400751,85 @@ created_by: dph
 creation_date: 2017-04-28T12:39:13Z

 [Term]
+id: GO:0061868
+name: hepatic stellate cell migration
+namespace: biological_process
+def: "The orderly movement of a hepatic stellate cell from one site to another."␣
↪[PMID:24204762]
+intersection_of: GO:0016477 ! cell migration
+intersection_of: results_in_movement_of CL:0000632 ! hepatic stellate cell
+created_by: dph
+creation_date: 2017-05-01T13:01:40Z
+
+[Term]
 id: GO:0065001
 name: specification of axis polarity
 namespace: biological_process
~/repos/MY-ONTOLOGY/src/ontology(issue-13390) $
```

See Daily Workflow section for commit, push and merge instructions.

CHAPTER 7

Synonyms

A synonym indicates an alternative name for a term. Terms can have multiple synonyms.

7.1 The scope of a synonym may fall into one of four categories:

7.1.1 Exact

The definition of the synonym is exactly the same as primary term definition. This is used when the same class can have more than one name.

For example, hereditary Wilms' tumor has the exact synonoym familial Wilms' tumor.

Additionally, translations into other languages are listed as exact synonyms. For example, the Plant Ontology list both Spanish and Japanese translations as exact synonyms; e.g. anther wall has exact synonym 'pared de la antera' (Spanish) and ' '(Japanese).

7.1.2 Narrow

The definition of the synonym is the same as the primary definition, but has additional qualifiers.

For example, pod is a narrow synonym of fruit.

Note - when adding a narrow synonym, please first consider whether a new subclass should be added instead of a narrow synonym. If there is any uncertainty, start a discussion on the GitHub issue tracker.

7.1.3 Broad

The primary definition accurately describes the synonym, but the definition of the synonym may encompass other structures as well. In some cases where a broad synonym is given, it will be a broad synonym for more than one ontology term.

For example, Cyst of eyelid has the broad synonym Lesion of the eyelid.

Note - when adding a broad synonym, please first consider whether a new superclass should be added instead of a broad synonym. If there is any uncertainty, start a discussion on the GitHub issue tracker.

7.1.4 Related

This scope is applied when a word of phrase has been used synonymously with the primary term name in the literature, but the usage is not strictly correct. That is, the synonym in fact has a slightly different meaning than the primary term name. Since users may not be aware that the synonym was being used incorrectly when searching for a term, related synonyms are included.

For example, Autistic behavior has the related synonym Autism spectrum disorder.

7.2 Synonym types

Synonyms can also be classified by types. The default is no type. The synonym types vary in each ontology, but some commonly used synonym types include:

- abbreviation - to indicate the synonym is an abbreviation. **Note** the scope for an acronym should be determined on a case-by-case basis. Not all acronyms are necessarily exact.
- ambiguous - to indicate the synonym is open to more than one interpretation; may have a double meaning
- dubious synonym - to indicate the synonym may be suspect
- layperson term - to indicate the synonym is common language (used by the Human Phenotype Ontology)
- plural form - indicating the form of the term that means more than one
- UK spelling - the english language spelling that is used in the United Kingdom (UK) but not in the United States (US)

7.3 Database cross references

Whenever possible, database cross-references (dbxrefs) for synonyms should be provided, to indicate the publication that used the synonym. References to PubMed IDs should be in the format PMID:XXXXXXX (no space). However, dbxrefs for synonyms are not mandatory in most ontologies.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 8

Deleting Asserted Subclasses

Asserted is_a parents do not need to be retained as entries in the 'SubClass of' section of the Description window in Protege if the logical definition for a term results in their inference.

If you have created a logical defintion for your term, you can delete the asserted is_a parent by clicking on the X to the right term.

Once you synchronize the Reasoner, you will see the reasoned classification of your new term, including the inferred is_a parent(s).

If the inferred classification does not contain the correct parentage, or doesn't make sense, then you will need to modify the logical definition.

If an existing term contains a logical definition and still shows an asserted is_a parent in the 'SubClass of' section, you may delete that asserted parent, as well. Just make sure to run the Reasoner to check that the asserted parent is now replaced with the correct reasoned parent(s).

See also [GO Editors Guide on Obsoletion])http://wiki.geneontology.org/index.php/Obsoleting_an_Existing_Ontology_Term)

CHAPTER 9

Obsoleting an Existing Ontology Term

See Daily Workflow for creating branches and basic Protégé instructions.

1. Check if the term (or any of its children) is being used for annotation:
    - Go to AmiGO, search for the term, either by label or ID
    - Use filters on the left to look at direct annotations, EXP annotation, InterPro2GO annotations
    - Notify affected groups
2. Check if the term is used elsewhere in the ontology
    - In Protégé, go to the 'Usage' tab to see if that ID is used elsewhere. Search for the term name or the term IRI (ie with underscore between GO and the numerical part of the ID, for example: '"GO_0030722"'
    - If the term is a parent to other terms or is used in logical definitions, make sure that another term replaces the obsolete term
3. Send a notification email. Template:
    - SUBJECT: Proposal to obsolete [GO:ID] [GO term name]
    - BODY: Dear all, The proposal has been made to obsolete: [GO:ID] [GO term name]. The reason for obsoletion is [SPECIFY]. There are X experimental annotations to this term. There are X InterPro2GO mappings to this term. Any comments can be added to the issue: [link to GitHub ticket]. We are opening a comment period for this proposed obsoletion. We'd like to proceed and obsolete this term on [7 days after the message; unless it involves a lot of reannotation, in this case it can be longer] *** Unless objections are received by [DATE] , we will assume that you agree to this change. ***

Remember to list the databases affected by the obsoletion and tag people in the GH ticket Check go-slims

Possible reasons for obsoletions:

- The reason for obsoletion is that there is no evidence that this function/process/component exists. (eg: GO:0019562 L-phenylalanine catabolic process to phosphoenolpyruvate; GO:0097605 regulation of nuclear envelope permeability'; GO:0015993 molecular hydrogen transport)
- The reason for obsoletion is that the term is not clearly defined and usage has been inconsistent (eg: GO:0030818 negative regulation of cAMP biosynthetic process)

- The reason for obsoletion is that this term represent a GO-CAM model. (eg: GO:0072317 glucan endo-1,3-beta-D-glucosidase activity involved in ascospore release from ascus; GO:0100060 negative regulation of SREBP signaling pathway by DNA binding)
- The reason for obsoletion is that this term represent an assay and not a real process. (eg: GO:0035826 rubidium ion transport)
- The reason for obsoletion is that the data from the paper for which the term was requested can be accurately described using [appropriate GO term]. (eg: GO:0015032 storage protein import into fat body)
- etc

OBSOLETION PROCESS

1. Navigate to the term to be obsoleted.
2. Make the status of the term obsolete:
    1. In the 'Annotations' window, click on the + sign next to 'Annotations'.
    2. In the resulting pop-up window, select `owl:deprecated` from the left-hand menu.
    3. Make sure the 'Literal' tab view is selected from the right-hand tab list. Type `true` in the text box.
    4. In the 'Type' drop-down menu underneath the text box, select `xsd:boolean`
    5. Click OK. You should now see the term crossed out in the Class hierarchy view.
3. Remove equivalence axiom: In the 'Description' window, under the 'Equivalent To', click the `x` on the right-hand side to delete the logical definition.
4. Remove 'SubClass Of' relations: In the 'Description' window, under the 'SubClass Of' entry, click the `x` on the right-hand side to delete the SubClass Relation.
5. Add 'obsolete' to the term name: In the 'Annotations' window, click on the `o` on the right-hand side of the rdfs:label entry to edit the term string. In the resulting window, in the Literal tab, in front of the term name, type: `obsolete` For example: `obsolete gamma-glutamyltransferase activity` **Note the case-sensitivity. Make sure to have a space (and no other character) between 'obsolete' and the term label**.
6. Add 'OBSOLETE' to the term definition: In the 'Description' window, click on the `o` on the right-hand side of the definition entry. In the resulting window, in the Literal tab, at the beginning of the definition, type: `OBSOLETE.` For example: `OBSOLETE. Catalysis of the reaction: (5-L-glutamyl)-peptide + an amino acid = peptide + 5-L-glutamyl-amino acid.` **Note the case-sensitivity**.
7. Add a statement about why the term was made obsolete: In the 'Annotations' window, select + to add an annotation. In the resulting menu, select `rdfs:comment` and select Type: `Xsd:string`. Consult the wiki documentation for suggestions on standard comments:
    - http://wiki.geneontology.org/index.php/Curator_Guide:_Obsoletion
    - http://wiki.geneontology.org/index.php/Obsoleting_GO_Terms
    - http://wiki.geneontology.org/index.php/Editor_Guide
8. If the obsoleted term was replaced by another term in the ontology: In the 'Annotations' window, select + to add an annotation. In the resulting menu, select `term replaced by` and enter the ID of the replacement term.
9. If the obsoleted term was not replaced by another term in the ontology, but there are existing terms that might be appropriate for annotation, add those term IDs in the 'consider' tag: In the 'Annotations' window, select + to add an annotation. In the resulting menu, select `consider` and enter the ID of the replacement term.
10. Save changes.

See Daily Workflow section for commit, push and merge instructions.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 10

Merging Ontology Terms

See Daily Workflow for creating branches and basic Protégé instructions.

Note Before performing a merge, make sure that you know all of the consequences that the merge will cause. In particular, be sure to look at child terms and any other terms that refer to the 'deprecated' term. In many cases a simple merge of two terms is not sufficient because it will result in equivalent classes for child terms. For example if deprecated term X is going to be merged into target term Y and 'regulation of X' and 'regulation of Y' terms exist, then you will need to merge the regulation terms in addition to the primary terms. You will also need to edit any terms that refer to the deprecated term to be sure that the names and definitions are consistent.

1. **Find the ID of the term in which the deprecated term will be merged**
   - Navigate to 'winning' term using the Search box. **Copy the ID of the winning term somewhere.**
2. **Remove annotations from the deprecated terms**
   - Navigate to the term to be deprecated.
   - Remove the logical definition by clicking on the `x` on the right.
   - Remove all subclasses by clicking on the `x` on the right.
   - Look at the definition; if it does not seem relevant, remove it by clicking on the `x` on the right; otherwise copy/paste it somewhere to refer to when reviewing the definition for the winning term.
   - Note down the created_by and created_date (there can only be one value per term for each of these fields; this will be useful if you need to pick one after the merge is done).
   - Check existing list of synonyms to see if they need to be moved to the new term, otherwise delete them by clicking on the `x` on the right.
3. **Change the ID of the term to be deprecated to the winning term's ID**
   - In the term to be deprecated, click on Refactor > Rename entity' in the Protege menu (shortcut: `command-U`)
   - Copy the ID of the winning term (obtained in Step 1).
   - Be sure to use the underscore **_** in the identifier instead of the colon **:**, for example: `GO_1234567`. Make sure that the 'change all entities with this URI' box is checked.

4. **Make the deprecated ID an 'alternative ID'**
    - Navigate to the winning term. In the Annotations box, locate the ID of the deprecated term. Click the `o` to change the ID type.
    - In the resulting pop-up window, making sure the 'Literal' tab is selected in the top right side box, select `has_alternative_id` from the list on the left side. Double check that the entry corresponds to the GO ID of the deprecated term.
    - Click 'OK'. The deprecated term identifier should now have the label `has_alternative_id` instead of `id`.
5. **Change deprecated term label to a synonym**
    - In the annotations box of the winning term there are now two terms with labels 'rdfs:label'. Click the `o` to change the label of the deprecated term.
    - In the resulting pop-up window, select the appropriate synonym label from the list on the left:
        1. `has_broad_synonym`
        2. `has_exact_synonym`
        3. `has_narrow_synonym`
        4. `has_related_synonym` (if unsure, this is the safest choice)
6. **Fix synonyms**
    - In the annotations box of the winning term, check the list of synonyms to see if they are all still make appropriate.
7. If needed, fix the definition, using information from the deprecated term as appropriate.
8. **Synchronize the reasoner** and make sure there are no terms that have identical definitions as a result of the merge. These are displayed with an 'equivalent' sign in the class hierarchy view on the left hand panel.
9. Save changes.

See Daily Workflow section for commit, push and merge instructions.

TROUBLESHOOTING: Travis/Jenkins errors

1. **Merging a term that is used as 'replaced by' for an obsolete term**

```
    :: ERROR: ID-mentioned-twice:: GO:0048126
      GO:0030722 :: ERROR: has-definition: missing definition for id
```

The cause of this error is that Term A (GO:0048126) was obsoleted and had replace by Term B (GO:0030722). The GO editor tried to merge Term B into a third term term C (GO:0007312). The Jenkins checkk failed because 'Term A replaced by' was an alternative_id rather than by a main_id. Solution: In the ontology, go to the obsolete term A and replace the Term B by term C to have a primary ID as the replace_by.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 11

Term Comments

11.1 Adding Comments to Terms

Comments may be added to ontology terms to further explain their intended usage.

Wherever possible, we strive to use standard language for similar types of comments and suggest alternative terms to use.

Some examples of comments, and standard language for their usage, are:

11.2 Do Not Annotate

This term should not be used for direct annotation. It should be possible to make a more specific annotation to one of the children of this term.

Example: GO:0006810 transport

Note that this term should not be used for direct annotation. It should be possible to make a more specific annotation to one of the children of this term, for e.g. transmembrane transport, microtubule-based transport, vesicle-mediated transport, etc.

11.3 Do Not Manually Annotate

This term should not be used for direct manual annotation. It should be possible to make a more specific manual annotation to one of the children of this term.

Example: GO:0000910 cytokinesis

Note that this term should not be used for direct annotation. When annotating eukaryotic species, mitotic or meiotic cytokinesis should always be specified for manual annotation and for prokaryotic species use 'FtsZ-dependent cytokinesis; GO:0043093' or 'Cdv-dependent cytokinesis; GO:0061639'. Also, note that cytokinesis does not necessarily result in physical separation and detachment of the two daughter cells from each other.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 12

Adding a term to a Subset (Slim)

See Daily Workflow for creating branches and basic Protégé instructions.

1. In Protege, navigate to the term you wish to add to a subset (slim).
2. With the term selected, click on the Entities tab.
3. In the Annotation window on the right, click on the + to the right of 'Annotations' at the very top of the window.
4. In the pop-up window that appears, click on `in_subset` on the left-hand panel.
5. In the right-hand panel click on the 'Entity IRI' tab.
6. Navigate to the 'Annotation Properties' sub-tab.
7. Navigate to the subtypes of `subset_property` and select the appropriate slim.
8. Click on OK to save your edits.

See Daily Workflow section for commit, push and merge instructions.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 13

Adding a new Subset (Slim)

See Daily Workflow for creating branches and basic Protégé instructions.

1. In the main Protege window, click on the Annotation Properties tab.
2. Navigate to the `subset_property` and select it.
3. Click on the top left-hand button of the window to add a new subset property.
4. In the pop-up window add the name of the new slim. The identifier will fill in according to your preferences and will be the next identifier in your set. Click on Refactor in the menu. Select rename entities.
5. Replace the `IDSPACE_ identifier` with the name of your new slim. It is standard to use the same string as when you created the term.
6. In the annotations tab, click on the `+`.
7. In the pop up window, select `rdfs:comment`.
8. In the right hand window, type a small descriptor statement for the slim. Select `xsd:string` as the type.
9. Click OK to save the changes. You should now see the comment field in the annotations tab.

See Daily Workflow section for commit, push and merge instructions.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 14

Adding taxon restrictions

See Daily Workflow for creating branches and basic Protégé instructions.

1. `Only in taxon` relations are added as `Subclasses`.
    1. Navigate to the term for which you want to add the only in taxon restriction.
    2. In the Description window click on the `+`.
    3. In the pop-up window type a new relationship (e.g. `'only in taxon' some Viridiplantae`).
    4. The taxa available are imported ontology terms and can be browsed just like any other ontology term.
2. `Never in taxon` relations added as `Annotations`.
    1. Navigate to the term for which you want to add the never in taxon restriction.
    2. In the class annotations window, click on the `+`.
    3. In the left-hand panel, select `never_in_taxon`.
    4. In the right-hand panel, in the Entity IRI tab, navigate to the correct taxon. The full path is: thing/continuant/independent continuant/material entity/object/organism.
    5. Select the appropriate taxon.
    6. Click OK to save your changes.

See Daily Workflow section for commit, push and merge instructions.

NOTE This documentation is incomplete, for now you may be better consulting the GO Editor Docs

CHAPTER 15

Adding Terms to the Import Files

Terms are imported to GO from other ontologies, but not all terms from external ontologies are imported. Occasionally, you will find that a valid identifier exists in an external ontology, but the identifier is not available in Protege because that term is not yet imported. To import a term from an external ontology:

1. Navigate to the imports folder on GitHub, located at https://github.com/geneontology/go-ontology/tree/master/src/ontology/imports.
2. Look in the list of ontologies for the ontology that contains the term you wish to import.
3. Identify the `ontology_terms.txt` file for the target ontology. For example, for the addition of a new taxon, the file can be found at https://github.com/geneontology/go-ontology/blob/master/src/ontology/imports/ncbitaxon_terms.txt.
4. Click on the icon of a pencil in the upper right corner of the window to edit the file.
5. Add the new term on the next available line at the bottom of the file.
6. Click preview changes.
7. You can now either commit the file directly to master or create a branch and a pull request as described before.