VocBench – Quick guide for classification maintainers

FAO - ESS

Last update Nov 4, 2020

CONTENTS

1. Overview and Initial steps......................................................................................................................... 3
   1.1 – General overview ................................................................................................................................. 3
       Data model ............................................................................................................................................... 3
       Vocbench interface ................................................................................................................................. 8
   1.2 – Creation of a project ............................................................................................................................ 13
   1.3 – Creation of a classification (“scheme”) .............................................................................................. 13
       Checking the data model ......................................................................................................................... 13
       Creating a new classification (“scheme”) ............................................................................................... 14
       Adding metadata .................................................................................................................................. 15
       Scheme and classification/dataset metadata ......................................................................................... 17
2. Population and update of a classification using the Vocbench editing interface........................................ 21
   2.1 Population ........................................................................................................................................... 21
       Adding a new term .................................................................................................................................. 21
       Adding metadata .................................................................................................................................. 23
       Putting terms into a hierarchy .................................................................................................................. 25
       Adding translations ................................................................................................................................. 26
       Adding correspondences ......................................................................................................................... 28
   2.2 Update .................................................................................................................................................. 31
3. File-based population and update of a classification .................................................................................. 31
   3.1 Population ........................................................................................................................................... 31
       From an RDF file .................................................................................................................................... 32
       From an Excel / CSV file .......................................................................................................................... 32
   3.2 Update .................................................................................................................................................. 39
4. File-based population and web-based update ............................................................................................. 40
5. ADVANCED: Custom data models ............................................................................................................. 41
5.1 Identifying / designing the custom data model............................................................................................................. 41
   Identifying an appropriate schema or ontology .................................................................................................................. 42
   Creating a new schema or ontology .................................................................................................................................. 42
   Importing a schema or ontology in the Vocbench .................................................................................................................. 43
5.2 Importing a classification with a custom data model ...................................................................................................... 47
   5.2.1 Identifying and importing a suitable schema / ontology .......................................................................................... 48
   5.2.2 Importing the concepts ................................................................................................................................................. 51
6. Versioning .................................................................................................................................................................................. 61
7. Roles and permissions in the Vocbench .................................................................................................................................. 62
8. Publication .................................................................................................................................................................................... 63
1. OVERVIEW AND INITIAL STEPS

1.1 – General overview

Data model

SKOS

The Vocbench already has an underlying core data model that fits most classifications, the SKOS¹ data model. This model fits most classifications because it was designed for Knowledge Organization Systems (KOS). Wikipedia defines KOS: “Knowledge Organization Systems (KOS), concept system or concept scheme is a generic term used in knowledge organization about authority files, classification schemes, thesauri, topic maps, ontologies etc.” Statistical classifications are classification schemes and in most cases they have a simple data model. The terms in a classification normally have a code, a name/label in one or more languages, possibly alternative names, definitions, explanatory notes or editorial notes, hierarchical relations and possibly correspondences with other terms in other classifications. The classification “container” normally just has a name, perhaps an acronym, and may need some metadata like description, publisher, dates etc.

Typical data model for a simple classification

<table>
<thead>
<tr>
<th>term</th>
<th>classification</th>
<th>subset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Acronym / ID</td>
<td>Acronym / ID</td>
</tr>
<tr>
<td>Name (en, fr, es...)</td>
<td>Name (en, fr, es...)</td>
<td>Name (en, fr, es...)</td>
</tr>
<tr>
<td>Alternative name (en, fr, es...)</td>
<td>Description</td>
<td>Classification</td>
</tr>
<tr>
<td>Note</td>
<td>Publisher</td>
<td>Subset</td>
</tr>
<tr>
<td>Definition</td>
<td>... other metadata</td>
<td>... other metadata</td>
</tr>
<tr>
<td>Parent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correspondences</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial correspondences</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Core SKOS model

On orange background: classes
Black on white background: properties

Since the Vocbench is built on the SKOS model, the Vocbench interface uses SKOS terminology: the

following basic correspondences between common classification terminology and SKOS terminology should be enough to find one’s way around the Vocbench interface:

<table>
<thead>
<tr>
<th>COMMON TERMINOLOGY</th>
<th>SKOS TERMINOLOGY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification → Term</td>
<td>URI (unique http:// identifier, non-local ID)*</td>
</tr>
<tr>
<td>Code / acronym → Name / description</td>
<td>Concept Scheme / Scheme</td>
</tr>
<tr>
<td>Note / explanatory note → Definition</td>
<td>Concept</td>
</tr>
<tr>
<td>Definition → Parent</td>
<td>notation</td>
</tr>
<tr>
<td>Parent → Child</td>
<td>preferred label (prefLabel)</td>
</tr>
<tr>
<td>Correspondences → Partial correspondences</td>
<td>note / scope note</td>
</tr>
<tr>
<td>Partial correspondences → Subset</td>
<td>definition</td>
</tr>
<tr>
<td>Subset → Column name / variable</td>
<td>broader</td>
</tr>
<tr>
<td></td>
<td>narrower</td>
</tr>
<tr>
<td></td>
<td>exact matches (exactMatch)</td>
</tr>
<tr>
<td></td>
<td>close / broad / narrow matches (closeMatch)</td>
</tr>
<tr>
<td></td>
<td>Collection</td>
</tr>
<tr>
<td></td>
<td>Property</td>
</tr>
</tbody>
</table>

* URI: This is not exactly a property: it’s the way each “thing” is uniquely identified in RDF. The Vocbench is built on an RDF architecture, so all “things” (in our case, concepts, schemes, collections) have URIs and relations between different things are represented as relations between their URIs. All schemes and concepts created in the Vocbench have a URI (either automatically generated or using a specific local identifier of your choice).

These classes and properties are displayed in the Vocbench using their RDF representation: prefix:class/property, where the prefix stands for the vocabulary from which the class/property comes, in this case SKOS (conventionally, classes start with a capital letter, properties with a lowercase letter). So you will see in the Vocbench that a concept is of type skos:Concept, its preferred label is in a property called skos:prefLabel etc.

### Standard terminology

Although the class and property names above can look less obvious than the familiar terminology, consider the key advantage that using the SKOS terminology and model present: this is a standard “vocabulary” (a set of agreed terms to express a data model). Everybody familiar with semantic technologies knows what skos:prefLabel or skos:broader means, and so do software applications that are aware of semantic technologies. (So for instance, any semantic application would represent skos:broader relationships in a tree, without the need for the programmer to read documentation about the data structure.)

Typically, for classifications that have codes, labels in one or more languages, definitions in one or more languages, explanatory notes in one or more languages, editorial/use notes, a
tree structure and correspondences with terms in other classifications, the core SKOS model is enough.²

A graph-like representation of the SKOS RDF model is shown below:

![SKOS RDF Model Diagram]

Additional vocabularies

In the Caliper instance of the Vocbench, we added also support for additional vocabularies that help manage statistical classifications:

- **XKOS**
  The SKOS extension for representing statistical classifications³ models some additional features of statistical classifications, in particular: a) it allows to group correspondences (called Concept Associations) under sets of Correspondences (each set represents a set of correspondences between two classifications); b) it allows to represent hierarchical levels (e.g. section, division, group, class, subclass) and associate concepts with the level to which they belong; c) it provides some additional properties to represent relations between concept schemes (e.g. between versions – follows, supersedes – or variants).
  So you will see in the Vocbench that the CPC 2.1 “concept scheme” xkos:follows the CPC 2.0 concept scheme, and if you go under Collections you will see that CRS has 5 xkos:HierarchicalLevel: DAC code 1st, DAC code 2nd, CRS code, voluntary code, and for each level you can see the relevant codes (skos:member).

- **DCAT**
  The W3C DCAT vocabulary⁴ has been added primarily to manage the metadata of both a) the actual classifications (the content per se, independent of the representation) and b) the SKOS

---

² The OWL vocabulary is also loaded in the Vocbench, so all the OWL properties can be used (more advanced, mainly applicable to the data model and the scheme). For a complete overview of the properties available in the Vocbench for the three main classes (Concept, ConceptScheme and Collection), see the SKOS vocabulary ([https://www.w3.org/TR/swbp-skos-core-spec/#propertiesTitle](https://www.w3.org/TR/swbp-skos-core-spec/#propertiesTitle)). For advanced properties to describe ontological aspects, you can also see the OWL vocabulary ([https://www.w3.org/TR/2004/REC-owl-guide-20040210/#TermIndex](https://www.w3.org/TR/2004/REC-owl-guide-20040210/#TermIndex)).
³ [https://ddialliance.org/Specification/XKOS/1.2/OWL/xkos.html](https://ddialliance.org/Specification/XKOS/1.2/OWL/xkos.html)
⁴ [https://www.w3.org/TR/vocab-dcat-2/](https://www.w3.org/TR/vocab-dcat-2/)
RDF representation and the relations between the two (e.g. the fact that the CPC 2.1 RDF/SKOS serialization is a “distribution” – dcat:distribution - of the CPC 2.1 classification).

In the implementation of the DCAT model, we consider the actual classification as a dcat:Dataset and the SKOS serialization as a dcat:Distribution. So metadata about the classification (the official title, the original publisher, the version of the classification, the date of publication, the web page etc.) is associated with the Dataset entity, while metadata about the SKOS serialization (the name of the SKOS version, the publisher of the SKOS – Caliper -, the version of the SKOS etc.) is associated with the Distribution entity. IMPORTANT: a dataset is defined by its content and its authors/publishers, so a variant of a classification with even just one added or removed code is a different dataset: this is why in most cases there is a one-to-one relation between dataset and SKOS scheme / distribution.

DCAT includes properties from the Dublin Core Terms vocabulary for basic resource metadata, so the dct:title, dct:creator, dct:publisher, dct:source, dct:issued, dct:license have been used for both the metadata of the Dataset and the metadata of the Distribution.

To represent relations between datasets, we use the following properties when appropriate: dct:versionOf, prov:wasDerivedFrom, xkos:follows, xkos:variant.

**Licensing and provenance**

Licensing metadata is implemented using dct:license at the level of the dcat:Dataset (the original classification) and at the level of the dcat:Distribution (the SKOS serialization):

- for the classification, dc:license (DC instead of DCT is used for technical reasons) only contains a general sentence saying "Dataset license to be checked with the dataset creator:" followed by the URL of the source classification page.
- for the distribution, dct:license points to a page on the Caliper website describing the license for all distributions published by Caliper (a placeholder for now, waiting for an official license): [http://stats-class.fao.uniroma2.it/caliper/caliper-distribution-license](http://stats-class.fao.uniroma2.it/caliper/caliper-distribution-license).

In addition, for the distribution, dct:provenance points to a page on Caliper describing the provenance chain for all distributions published by Caliper (a placeholder for now, pointing also to the Sources page of the Caliper website) [http://stats-class.fao.uniroma2.it/caliper/caliper-provenance-statement](http://stats-class.fao.uniroma2.it/caliper/caliper-provenance-statement).

- **PROV**

The PROV (Provenance) Ontology (PROV-O)\(^5\) is used to allow for better description of provenance relations between classifications. In particular, we use the prov:wasDerivedFrom property for variants and other properties can be used when appropriate: prov:wasInfluencedBy, prov:wasQuotedFrom, prov:wasRevisionOf, prov:hadPrimarySource, prov:alternateOf, prov:specializationOf.

**Subsets, extensions, variants**

In cases where it’s necessary to represent subsets (e.g. the CPC 2.1 terms relevant for Fisheries) or extensions (subsets+additions) (e.g. the CPC 2.1 Expanded for Agriculture) or variants of a classification (e.g. different variants of the M49), three approaches are used:

---

\(^5\) [https://www.w3.org/TR/prov-o/](https://www.w3.org/TR/prov-o/)
a) **Flat subsets** with no hierarchy: they’re represented as SKOS Collections belonging to the original scheme. (SKOS collections are conceived indeed to represent subsets, but only flat ones, as they do not support hierarchies well.)

To link collections to a specific scheme, we use the SKOS inScheme property. Collections have the skos:member property to link to all concepts belonging to the collection.

Example of a flat subset treated as a Collection: the CPC 2.1 terms relevant for Fisheries.

b) **Extensions**: the main standard classification is the main concept scheme, while extensions are separate concept schemes, all sharing the common concepts with the main classification and having some additional concepts. Concepts that are common to the main classification and to all extensions belong to all concept schemes; other concepts may belong to only one or two. The skos:inScheme property of concepts is used to link concepts to the scheme to which they belong.

The original datasets are related through the prov:wasDerivedFrom property.

Examples of extensions treated as separate schemes: the CPC 2.1 Expanded for agriculture and the CPC 2.1 expanded for fertilizers. (For example, the concepts in the original CPC 2.1 that also belong to the CPC 2.1 Expanded for ag have their skos:inScheme property set to both the original CPC 2.1 scheme and the CPC 2.1 Expanded for ag scheme; and the CPC 2.1 Expanded for Ag scheme has the prov:wasDerivedFrom property set to the CPC 2.1 scheme.)

c) **Variants** of the same classification: separate schemes that share the common concepts. All concepts have the same base URI (the base URI of the main scheme) but belong only to the schemes / variants in which they’re present.


The same approach may be applied in the near future to the variants of the CPC 2.1 Expanded for Ag used in SWS, once the key variants are identified.

**Relations**

- To represent relations between datasets / classifications, we use the following properties when appropriate: dct:versionOf, prov:wasDerivedFrom and possibly other dct: or prov: properties like dct:isVersionOf, dct:hasVersion, dct:replaces, dct:isReplacedBy, dct:references, dct:isReferencedBy, dct:requires, dct:isRequiredBy; prov:wasDerivedFrom, prov:wasInfluencedBy, prov:wasQuotedFrom, prov:wasRevisionOf, prov:hadPrimarySource, prov:alternateOf, prov:specializationOf:

  `<dataset1> prov:wasDerivedFrom <dataset2>`

- To represent relations between schemes, we use the XKOS properties (but all dct: relation properties can be used with schemes as well):

  `<main_scheme> xkos:variant <variant_scheme>`

  `<new_scheme> xkos:follows/skos:supersedes <old_scheme>`

- To represent the relation between a classification/dataset and a scheme/distribution, we use the dcat:distribution property:

  `<dataset> dcat:distribution <scheme>`
To represent the belonging relation between a scheme and another resource (e.g. a dcat:Dataset or a classification family) we use the xkos:belongsTo property:

```xml
<scheme> xkos:belongsTo <resource>
```

As you may have noted, we treat both extensions and variants in a similar way, except for the relations between them and the main classification. This allows us to be flexible on the definition of a subset / extension / variant: they’re all separate schemes sharing concepts with their main classification, but if a code list stops being considered a variant and starts being considered an official extension, we only need to change the relations between datasets and between schemes (e.g. from prov:wasDerivedFrom to dct:versionOf or from xkos:variant to xkos:follows). At the moment, we consider extensions as agreed classifications, more or less formally published (e.g. the CPC 2.1 Expanded for Ag), and variants as small variations needed for domain-specific processing or internal purposes.

The diagram below expands the basic SKOS model with the DCAT and XKOS modelling of metadata for and relations between classifications / datasets and schemes / distributions:

---

**Vocbench interface**

To start working in the Vocbench, login on the homepage:

http://stats-class.fao.uniroma2.it/vocbench3/#/Home.

If you don’t have an account with special permissions, you can browse the content using the guest credentials: guest@guest.com / guest.
When you log into the Vocbench, you will see a list of all projects on which you have some type of permission (view, edit, manage). These may also be empty projects that still have to be populated. Just click one of the projects to start.

Let’s open for example the CPC 2.1 project (staging stands for a non-public instance, a project used to test changes before publication).

All operations around concepts and schemes are performed under the “Data” menu item of the Vocbench platform: if you click on Data and then on Concept you will see all the terms / concepts in the classification:
About data and metadata in the Vocbench

Statisticians are used to considering classifications as “metadata”. In the Semantic Web, all instances or “things” that are represented and described are “data”, so also classifications / concept schemes and terms / concepts are data, while the properties used to describe them (labels, codes, definitions) are their metadata. The classes and properties that you can use for representing your things are defined in “vocabularies” (set of agreed terms to express the data model for a specific type of thing).

In the Vocbench, under the Data menu item, you manage all data or “things” (concepts, schemes, collections, classes, properties) and the values of their metadata (in the right panel).

Under the Metadata menu item, you configure the vocabularies used to express the data model (SKOS, OWL and additional ones of your choice), and therefore the “metadata definition”.

(You probably won’t need to work under the Metadata tab if you just need the core SKOS model, which is already set up. Instructions on how to configure the Metadata for other cases are in chapter 5 on custom data models).

To see all the details (properties) of a concept, click on it in the left panel and the properties will appear in the right panel:
In the Vocbench, you will find the property names normally prepended with the skos: prefix (e.g. skos:prefLabel). Since additional vocabularies can be used in the Vocbench, the prefix always helps to remember from which vocabulary the property comes. *(In the future we can customize these properties to be displayed with other names of our choice.)*

For an initial tour, besides checking concepts and their properties, you can check the classification itself, the “scheme”: click on the Scheme tab to see the scheme and click on it to see its properties in the right panel:

**Scheme and dataset metadata**

You can see above that the metadata of the concept scheme is about the SKOS version of the classification (CPC 2.1 as RDF/SKOS...), not about the classification itself (CPC 2.1). This is a design choice, to allow us to describe separately the SKOS version and the original classification as a conceptual entity,
not linked to a specific representation / serialization. (See the DCAT model described in the first chapter).

If you want to see the metadata about the classification, you have to select the dcat:Dataset entity. You can do it in two different ways:

1) Under the Scheme tab, select the scheme, go under the Other properties section and look for the xkos:belongsTo property: that property links the scheme to the dataset of which it is a representation.

Double-click on the value of that property to see all the metadata about the classification dataset.
2) Under the Class tab, select dcat:Dataset under dcat:Resource and you will see all the instances of datasets in the box below the Class list. Click on a dataset instance to see all the metadata about the classification dataset.

This initial tour should help you familiarize with the terminology and the panels.

The next chapters describe the initial steps needed to create a classification. These are steps common to all workflows (whether the classification is created directly in the Vocbench editing interface or on a file and then uploaded). Then chapters 2 and 3 will illustrate the two types of workflow separately.

1.2 – Creation of a project

(For the moment this is performed by Tor Vergata, in the future permissions can be given to FAO staff, after a short training.)

A project can contain different classifications.

The advantage of having different classifications in one project is that the terms of all classifications will be visible from other classifications and easily linked (e.g. for correspondences). The caveat is that maintainers have to master the system and be well organized: without selecting a specific classification, users may see all terms at the same time, so maintainers need to be careful to always select the relevant classification before performing any operation. Also when importing classifications in a multi-scheme project it is important to specify the belonging scheme and the base URIs in the file to be imported.

Decisions on how to organize projects and what to put in each project should be guided by general guidelines (e.g. establishing that projects are organized around topics or types of classifications, or that projects are organized around permissions, so in one project we put classifications that are managed by the same people / group). In the initial stage, since the Vocbench is only used for publishing, each classification has been put in a dedicated project: in some cases, in the same project also extensions or variants of the main classification have been included (for instance, the CPC 2.1 Expanded for Agriculture is in the CPC 2.1 project, and both M49 UNSD and M49 FAO are in the M49 project).

1.3 – Creation of a classification (“scheme”)

Checking the data model

Before creating a classification, the maintainer has to make sure that the data model in the project fits the classification that is going to be created.

As we said, the Vocbench already has an underlying core data model that fits most classifications (the SKOS data model, see 1.1). If the core SKOS model fits the new classification that has to be created, there is no need for further modelling. You can go ahead with the instructions below. Otherwise, go to chapter 5 on custom data models.

Typically, for classifications that have codes, labels in one or more languages, definitions in one or more languages, explanatory notes in one or more languages, editorial/use notes, a
tree structure and correspondences with terms in other classifications, the core SKOS model is enough.

Creating a new classification (“scheme”)

When you log into the Vocbench, you will see a list of all projects on which you have some type of permission (view, edit, manage). In order to create a new scheme, you need to select a project on which you have “project manager” permissions.

In this manual, we will use as an example the FAO Commodity List (FCL).

To create a new scheme, click on the first icon under the Data menu item → Scheme tab:

That will open a popup for creating the new scheme:

The URI is composed by the base URI that was established when creating the project plus a local path / identifier of your choice. This can be a string (like an acronym) or a number. If you leave the URI local path empty, a random local path will be generated. **URIs, including local paths, don’t need to be human-readable or convey any meaning, they just identify “things” (concept schemes, concepts) univocally for all machine operations. Using a meaningful acronym is just a hint for humans.**

Remember what we said about the difference between scheme / distribution and dataset / classification: the scheme we are creating is the RDF representation of the classification, hence the name we are giving it.

Scheme created:
Adding metadata

Then you may want to describe this new scheme, providing some metadata (like a description, the publisher, or the date of publication etc.). When you click on the scheme in the left panel, you can add metadata in the right panel:

The icons for adding or editing metadata properties are on the right side of the panel:
The icons for adding properties look like this:

The icons for editing the values of properties look like this:

For instance, if you want to edit the name (preferred label) of the scheme or add a new label in another language, you can use the icons on the right as illustrated below:

Additional metadata properties
If you don’t see all the properties you need and you want to select an additional property from the list of properties available in the Vocbench, click on the blue icon besides the “Other properties” box:

Scheme and classification/dataset metadata

*(The setup for this can be left to the Caliper Vocbench maintainers, if the classification maintainers find the process too complex.)*

Remember the distinction we made in the first chapter about metadata for the concept scheme / distribution (the SKOS RDF representation of the classification) and the classification itself: the SKOS representation is a skos:ConceptScheme and a dcat:Distribution, while the classification is a dcat:Dataset.

So if you want to provide metadata about the SKOS scheme (a name, a description, when it was created, by whom), you use the properties in the right panel of the scheme. If you want to provide metadata about the classification itself, you need to implement the DCAT model, which means that you have to first declare this scheme also a dcat:Distribution, then create a new dcat:Dataset and then use its dcat:distribution property to link to the scheme / distribution.

First, let’s tell Vocbench that the scheme we just created is also a dcat:Distribution: click on the plus sign besides “rdf:type” to declare the scheme also of an additional type:
The scheme is now also a dcat:Distribution:

Then, to create a new dataset, go to the Class tab, select the dcat:Dataset class under dcat:Resource and click on “Create instance”:

Then provide the final part of the URI (guidelines on URI conventions may follow), for instance if you’re creating the FCL dataset you can use fcl_dataset. (Remember that URIs are conventional, they do not need to convey meaning, but it may be useful for other users to use meaningful URIs.)
Now leaving dcat:Dataset selected under the Class tab you will have the new dataset instance below and you can select it to see its properties:

You can add a name to the dataset adding a “lexicalization” and selecting skos:prefLabel in the next popup:

Here you can provide the official name of the classification:
Then, in order to link this classification / dataset with its representation in SKOS (the scheme), add the dcat:distribution property under Other properties:

Choose dcat:distribution in the popup:

Select the distribution:

In the above popup, when you click on dcat:Distribution, all the dcat:Distribution entities will be displayed on the right (this is why we declared the scheme also as a dcat:Distribution): if the scheme has the same URI as the project, you will just see the colon symbol (:) to indicate the entity (if you go over it, you will see the URI). Select it and click on OK.

Now you have set up the dcat:Dataset and its relation with the scheme through the dcat:distribution property. You can add additional metadata about the dataset using again the “Add a property value”
button and adding for instance dc:creator and dc:publisher for the institution that created the classification, dct:title for its title (we used skos:prefLabel above, because the Vocbench privileges the SKOS lexicalizations, but dct:title is more suitable for the DCAT model), dct:source for a web page that gives access to the original file of the classification, dcat:landingPage for a web page that gives more information about it etc.

Additionally, to reversely link the scheme / distribution to the dataset (useful for navigating from the scheme to the dataset, and also for the final machine-readable RDF), you can go again under Scheme, select the scheme and use the xkos:belongsTo property to select the dataset of which this scheme is a representation:

Then select xkos:belongsTo and in the following popup select the dcat:Dataset instance.

The right editing panel with all the options to add and edit properties will be illustrated in more detail in chapter 2 on web-based population.

Once the scheme has been created (and optionally described), its population and maintenance can be performed either interactively on the web interface, using the Vocbench editing forms, or by uploading RDF or CSV / Excel files following a specific template.

The next two chapters illustrate the two different approaches.

2. POPULATION AND UPDATE OF A CLASSIFICATION USING THE VOCBENCH EDITING INTERFACE

2.1 Population

In this chapter we assume you have permissions to create concepts without the need for approval (you have “validator” permissions), so your changes are immediately effective. If you don’t have validator permissions, your changes will appear as temporary until a validator approves them. (See chapter 7 on Roles and permissions.)

Adding a new term

After creating the scheme, you start adding the terms of the classifications (in SKOS language: concepts).
Label and language: labels are translatable, so when you first create a term you need to specify which language you’re using for the label.

URI: Again, as for the scheme, the URI is composed by the base URI established when creating the project and a local path / identifier of your choice. Here you have to provide the local part of the URI. This can be a string (like an acronym) or a number. The recommended practice is to use the term code, as it is assumed to be unique. If you leave the URI local path empty, a random local path will be generated (URIs are conventional, they don’t need to be human-readable or convey any meaning, they just identify the concept univocally for all machine operations).

Schemes: here you select the scheme to which the concept belongs. In projects where you only have one scheme, this is straightforward. When you have more than one scheme, you can assign the concept to one or more schemes (e.g. if it belongs to different variants). You can always assign the concept to new schemes later.

New concept created:
Adding metadata

After creating the concept, additional metadata fields (“properties”) can be filled in the right panel (code, translations, notes, correspondences...).

Assigning a code

The most important thing to add is of course the code, which in SKOS terminology is the “notation” (a string notation used as a local identifier).

In an empty Vocbench project, you will probably not find the skos:notation property already available in the right panel, so you have to add a new property from the list of properties available in Vocbench:

Clicking on the blue icon will open the popup for selecting the additional property:

Select skos:notation and click on OK; then select typedLiteral and in the final popup insert the code:
Now, you will see the concept with its code in the left panel:

Adding other properties for a concept follows the same procedure.

**Some common properties are already available in the right panel**, like the Notes properties. In SKOS, there are different types of notes you can use to annotate your concept. These types of notes go from a definition of the concept to an example to a history/change indication.

For instance, if you want to add a definition, add a new note:

The following popup opens, with the different types of notes available:
- definition is the best property for an official definition of the concept
- changeNote and historyNote can be used to document changes between versions
- editorialNote is useful for indications related to the editorial workflow on the concept
- scopeNote is useful for indications related to the use of the concept
- example is useful for providing examples

Select skos:definition and click on OK, then select typedLiteral and click on OK and in the final popup type the text of the definition:

If you then want to add a scope note, add a new note again, select skos:scopeNote and proceed as above.

Putting terms into a hierarchy

You can add the parent / child relation (in SKOS terminology, broader / narrower relation) once you have inserted both the parent and the child.

So in the example of the FCL above, once you have created the parent terms, you can assign them as broaders to the narrower terms.

Click on the child / narrower concept in the left panel (e.g. in our example: 0015 – Wheat):

then look for the “Broaders” section in the right panel:
Click on the blue round icon besides Broaders to open the broader selection popup:

Select the broader term and click on OK.

The broader relation is added: check the hierarchy in the left panel.

Other operations you may need to perform when creating a classification (or more commonly later, in an incremental way) are adding translations and adding correspondences to other concepts in other classifications.

**Adding translations**

Literal properties can be translated in different languages. A typical literal property that needs to be translated is the label of the term.
In the example above, we created the “Wheat” term as an English label. Let’s say we want to add the Spanish translation. In the right panel, we look for the skos:prefLabel property under the Lexicalization box. We can add a translation in another language by clicking on the + sign on the right of the skos:prefLabel property:

And then select the language and provide the translation in the popup:

After you’ve added translations, you’ll see them all in the left panel:

and in the right panel:

Then you can edit and delete translations:
Adding correspondences

The Vocbench is built on an RDF architecture: in RDF, all “things” (in our case, concepts, schemes, collections) have URIs and relations between different things are represented as relations between their URIs. All terms created in the Vocbench have a URI (as we saw, either automatically generated or using a specific local identifier of your choice).

There are two ways of adding correspondences in the Vocbench:

- **Correspondences with concepts in the same project** (or in projects for which you have read permissions): in this case, the mapped concepts can be selected from a list: the relation with the URI of the mapped concept is implemented by the Vocbench behind the scenes.
- **Correspondences with concepts outside of the project**: in this case, the URI has to be provided manually. This means that the classifications to which the mapped terms belong need to be published somewhere with URIs. This is a requirement for semantic relations. If you want to provide correspondences with other terms using codes or other literal values, see how to add new properties for corresponding codes in chapter 5 on custom data models.

You can add correspondences in two ways (the second one is highly recommended):

1) In the same way as you do with other properties (e.g. as we did with skos:notation), but that way works well with option 1 (concepts in the same project) while it is a little more cumbersome for terms outside of the project.

2) Using the dedicated alignment procedure.

So let’s use the dedicated procedure for mapping, which is called “alignment”: under Data > Concept select the concept you want to map (or “align”) with another one, then in the right panel click on the dropdown arrow on the right and select “**Align with external resource**”: 
In the alignment popup choose the appropriate property for your type of mapping:

![Add new alignment](image)

SKOS has several types of mapping relations, which represent different degrees of mapping precision:

- **exactMatch**: this is a perfect 1-1 correspondence
- **closeMatch**: this is a generic close correspondence, with no indication of whether one of the concepts is broader or narrower (partial correspondence)
- **broadMatch**: a partial correspondence with a term that is broader in meaning
- **narrowMatch**: a partial correspondence with a term that is narrower in meaning

If the mapping target term is in your project, click on the pencil button and select “Browse local projects”:

![Add new alignment](image)

A popup will display all local projects and you can select concepts in those projects.

If the mapped concept is NOT in your project, you have to know the URI of that concept.

Let’s say that you have looked for similar concepts in the CPC 2.0 (and the CPC 2.0 is not in your project) and you have identified the following two terms as narrow (partial) correspondences:
Wheat, seed
Wheat, other

In Caliper, browsing the CPC 2.0 (http://stats-class.fao.uniroma2.it/skosmos_demo/cpc20/en/page/0111), you find that those concepts have the following URIs:

Wheat, seed: http://stats-class.fao.uniroma2.it/cpc/cpc_v2.0/01111
Wheat, other: http://stats-class.fao.uniroma2.it/cpc/cpc_v2.0/01112

In this case, you can select the skos:narrowMatch property in the alignment popup and then click on the pencil button and select “Enter value manually”:

And enter the URI of the mapped concept in the following popup:

After you add all the narrow matches, you will find them in the right panel:

Once all concepts have been created and arranged in a hierarchy and all desired metadata is added, the classification can be published (see chapter 8 on publication).
2.2 Update

The typical operations that are performed in update-mode, in maintenance, are:

- Commonly:
  - addition of translations;
  - addition of correspondences to other concepts in other classifications.

- Rarely, depending on editorial policy:
  - changes to the classification metadata (description, publisher);
  - changes to existing concepts (e.g. notes);
  - addition of new concepts.

The possibility of performing these operations depends on editorial guidelines established by the classification maintainer. In most cases, once the classification has been created and published, it should not be altered except for adding translations or correspondences. The editorial workflow may for instance prescribe that if new concepts are needed a new version should be created (see chapter 6 on versioning).

The classification URI cannot change and the codes should not change.

The update operations are the same as the addition and editing operations that are performed when the classification is first created, before publication, as seen in chapter 2.1.

Every time a set of update operations is finished (and, if necessary, has been approved), the classification has to be published again (see chapter 8 on publication).

3. FILE-BASED POPULATION AND UPDATE OF A CLASSIFICATION

Important: once the classification / scheme object is created (see 1.2), it can be populated and updated via file uploads. This chapter covers the typical case of classifications for which the core SKOS model is suitable: classifications that have codes, labels in one or more languages, alternative labels, definitions, explanatory notes, use notes, correspondences with other terms (as URIs) and a hierarchical structure.

If the core SKOS data model doesn’t fit the new classification, see chapter 5 on custom data models and in particular the dedicated chapter on “Data import with a custom data model”.

3.1 Population

After creating the classification (see 1.2), instead of creating the terms and the hierarchy manually, you can upload the content of the classification, if you have it serialized in:

- an RDF file
- an Excel / CSV file with a suitable structure
From an RDF file

If you created the classification using an RDF-based tool using the SKOS model, or if you converted it into RDF, you can upload the RDF file into the Vocbench project by clicking on the top-right “Global Data Management” menu and then on “Load data”:

![Global Data Management menu](image)

In the Load Data page, just browse and select the local file and the corresponding format:

![Load Data page](image)

Then click on Submit at the bottom of the page. If the RDF is correct and the necessary SKOS objects (the concepts, or optionally the collections) are included, if you go back under Data you will find the content of the classification there.

This import procedure normally doesn’t require any configuration or adjustment because the SKOS model is standard: any semantic tool that creates a SKOS RDF file creates it in the same way.

**NOTE:**

For IT people generating the RDF file: it is important that the RDF contains the statements that link the scheme with the top-level concepts (the `skos:hasTopConcept` property of the `skos:ConceptScheme`), otherwise you will have to fill this property manually under Data > Scheme, clicking on the scheme and adding top level concepts in the right panel under the `skos:hasTopConcept` box.

Until top-level concepts are linked to the scheme, in order to see the concepts you will have to un-select the scheme from under the Scheme tab.

Several classifications have already been converted to RDF using Python scripts and this can still be done in case the Excel/CSV import procedure below requires too much work on the files (writing a conversion script requires on average less than one day).

From an Excel / CSV file

If you want to upload a classification that has been created in an Excel file or is available in a CSV file, the Vocbench needs to understand the meaning of the column names, so that they can be converted into the correct SKOS properties.
The easiest way to do this is by adjusting your Excel file to use a specific template (below we provide links to a few typical templates). The template uses column names / headers that follow RDF standards that the Vocbench understands:

- The “code” column should be repeated twice: one column should be named skos:Concept, the other column skos:notation^^xsd:string (so that the code is used both for the local path for the concept URI and for the notation property)
- The columns for the names / labels of the concepts in one or more languages should have the following header: skos:prefLabel@xx (xx being the language prefix), e.g. skos:prefLabel@en, skos:prefLabel@fr etc.
- The columns that contain any other literal value that can be translated should follow the same pattern: skos:propertyValue@xx, e.g. skos:scopeNote@en, skos:definition@en
- The columns that contain values that represent relations (for instance parent/child or exact or close matches) should just be named with the name of the corresponding property: e.g. skos:broader for parent terms or skos:exactMatch for correspondences.
- You need to always include two columns:
  - skos:inScheme, with the same value for all concepts: the URI of the scheme to which the terms belong (the URI of the scheme you created before importing the terms: you can always find it in the Vocbench project under Data > Scheme: clicking on the scheme name the URI is the first thing you see at the top of the right panel)
  - skos:topConceptOf, which should contain the URI of the scheme ONLY if the concept is a top-level concept (here you may need a simple formula, see below)
- You can use the prefix (e.g. skos:) if you use properties from a vocabulary that is already in your project, otherwise you should use the full URI of the property. This will be necessary only if your data model goes beyond the SKOS model, in which case see chapter 5 on custom data models and in particular the dedicated chapter on “Data import with a custom data model”.

It is also important that the values in the rows are correct:

- **Literal values** (codes, labels, definitions) can just be left as they are. 
  *Note: it is ESSENTIAL that codes are written in the exact same way in all columns, especially in the skos:Concept column and the skos:broader column (no trailing spaces or missing leading zeros), otherwise the hierarchical relations will not be created correctly.*
- **URI values** (things and relations to things, like concepts, broader, narrower, matches) should always be in full form (the full URI), except for concepts that belong to the classification that is being imported: those can be simple codes (the Vocbench will automatically create URIs prepending the project base URI to the code).
  For instance, the skos:Concept column only includes concepts that belong to the classification, so you can just use the code and the Vocbench will convert it into a URI; the same thing is true for the skos:broader column, which indicates a child/parent relation with a term in the same classification. Instead, skos:exactMatch indicates a mapping relation with a term in a different classification, so the column should contain the full URIs of the mapped terms.
Download templates:

1) Template n. 1
   For classifications where the hierarchy is implemented through incremental codes, where the children have the same code as the parent plus one digit:
   [https://drive.google.com/file/d/1wU6pkcc15MPPK_Q3seRH494XJwqGCLZH/view?usp=sharing](https://drive.google.com/file/d/1wU6pkcc15MPPK_Q3seRH494XJwqGCLZH/view?usp=sharing)

2) ...

**Example: loading the CPC 2.1 classification from an Excel file**

First, go through the preliminary step of creating the new empty classification in your project in the Vocbench (1.1 and 1.2). In our example, we want to import the CPC 2.1 classification, so we create a new scheme with label CPC 2.1 and a URI of our choice (like [http://stats-class.fao.uniroma2.it/cpc/cpc_v2.1/](http://stats-class.fao.uniroma2.it/cpc/cpc_v2.1/)). Once you have the classification in place, you can import content.

The original format of the CPC 2.1 classification is the following:

<table>
<thead>
<tr>
<th>ID</th>
<th>CPC21code</th>
<th>CPC21title</th>
<th>CPC21ExplanatoryNote</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Agriculture, forestry and fishery</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>products</td>
<td></td>
</tr>
<tr>
<td></td>
<td>011</td>
<td>Cereals</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0111</td>
<td>Wheat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01111</td>
<td>Wheat, seed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01112</td>
<td>Wheat, other</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0112</td>
<td>Maize (corn)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01121</td>
<td>Maize (corn), seed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01122</td>
<td>Maize (corn), other</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0113</td>
<td>Rice</td>
<td></td>
</tr>
</tbody>
</table>

In this case we will apply template n. 1 downloadable above.

This file can be easily adapted to the SKOS template, but it doesn’t contain a column with the broader concept, so it will import all concepts with their codes and notes, but not the hierarchy. And it doesn’t contain the two mandatory columns with inScheme and topConceptOf.

Let’s first adjust the headers as indicated by the SKOS template:
Then, let’s add the skos:broader column, to implement the hierarchy in the Vocbench.

We have to create a column containing the corresponding broader terms: in the case of the CPC 2.1, the parent/child relation is quite simple: the parent term always has the same code as the child minus the last digit: so let’s apply a simple formula to the skos:broader column:

And copy the formula to all cells in the column.

(In other cases, hierarchical relations may be more complicated and you may have to do some manual editing or write a script.)

Then let’s add a simple column to tell Vocbench that all terms belong to the scheme we’ve just created: the SKOS property for this is skos:inScheme and as value we have to use the URI of the scheme we created in the Vocbench (in our example, we created the CPC 2.1 scheme with URI http://stats-class.fao.uniroma2.it/cpc/cpc_v2.1/):

Finally we need to add a column telling Vocbench if the term is a top concept of the scheme or not. For this, the SKOS property is skos:topConceptOf and the value is again the URI of the scheme. So we need to provide this value ONLY for the concepts that are first-level concept, which means the concepts that have no broader term. So we can use a simple IF formula: IF skos:broader is an empty string, write the scheme URI, otherwise write nothing:
And copy the formula to all cells in the column.

Now let’s import this file into our project in the Vocbench.

First, make sure you are in the right project and have created the CPC 2.1 scheme.

The Excel / CSV import procedure in the Vocbench is called Sheet2RDF and is available under Tools:

When you upload the file in the Sheet2RDF page, the content is displayed in the preview panel:
As you can see, all labels are green, which means the Vocbench understands them.

Then you need to execute two small operations that are normally useful only for developers (who may want to adjust the import code): you just execute them in order to get to the final import:

1) Click on the play icon to check the import script:

2) Then click on the play icon in the right panel to execute the import script (you don’t need to understand it):

And you will see a preview of the imported RDF in the bottom panel (again, you don’t need to understand it):
3) Now, if there are no errors, you can click on the import button in the bottom panel (the tooltip says “Add triples”: this is RDF language: it means add all the data):

Now all the new data has been imported, creating the hierarchy and setting the top-level concepts: check under Data > Concept:
If the core Vocbench data model doesn’t fit the new classification, see chapter 5 on custom data models and in particular the dedicated chapter on “Data import with a custom data model”.

3.2 Update

The procedure for updating a classification from a file is the same as the procedure for populating the classification from a file. However, attention should be paid to the type of update you want to perform: full overwriting or modification/increment.

**Full overwriting**

If you want the classification that is in the project to be completely deleted and the new import to fully replace it, do as follows:

- Delete all the content
  (DO NOT use the Clear Data function under Global Data Management for now: that removes also the vocabularies!!)
  Use a SPARQL query *(for the moment, this will improve)*: the text of the SPARQL query is:

    WITH : DELETE {?a ?b ?c} WHERE {?a ?b ?c}
Modification / increment

If you want to import a partial file of modifications or additional properties (e.g. a file of only translations or only correspondences) and add the new values to the existing classification in the Vocbench, you can just import the new file without deleting the current content, paying attention to some overwriting / duplication caveats:

- Properties whose values are URIs are not duplicated if the same URI value is imported again: so if your new file says for instance that concept 0111 has as parent concept 011 (which in Vocbench is transformed into a URI) and this relation had been already imported, the relation is simply overwritten by the second import and not duplicated
- Repeatable properties whose values are URIs (e.g. skos:broader and skos:exactMatch) can be used several times but with different values (if you import the same broader as above it won’t be duplicated, but if you import another broader it will be added)
- Repeatable properties whose values are literal (strings), like skos:prefLabel or skos:note will import any new value, even if the same string had already been imported, so if the second file you import has the same labels as the previous import, the labels will be duplicated.

In most cases, if the file contains the whole content of the classification, it is advisable to first delete the content of the scheme in the Vocbench. Otherwise, for incremental updates, it is better to upload files that contain only the new content.

4. FILE-BASED POPULATION AND WEB-BASED UPDATE

Another possible workflow is to populate the classification from a file initially and then update it using the Vocbench editing forms. This is useful for instance if you have a classification already in a file but you want to continue working on it using the specialized editing forms in the Vocbench, or at a certain point you want to add translations or correspondences directly in the Vocbench editing interface instead of working in Excel.

Being a combination of the two previously illustrated workflow, the procedures have already been described:
4.1 Population
See Chapter 3.1 on file-based population.

4.2 Maintenance
See chapter 2.2 on web-based maintenance

5. ADVANCED: CUSTOM DATA MODELS

5.1 Identifying / designing the custom data model

Again, consider that the SKOS/XKOS/DCAT model will probably cover 90% of the models of statistical classifications.

If the core data model of the Vocbench is not enough for your needs, you can “import” other data models. (In theory, you could even create a new data model in the Vocbench, but this may need ontologist skills. We will provide a quick example in this chapter anyway.)

It is common practice in RDF and Linked Data to reuse as much as possible data models that have already been published. Data models are formalized in “vocabularies” (agreed sets of terms and relations). The word vocabularies in the RDF community is quite general (they’re used for the definition of semantics) and can identify different types of agreed sets of terms:

1) Description / modelling vocabularies or schemas are the vocabularies that formalize the way certain things should be described and the relations they have with other things: they’re the most appropriate vocabularies to design data models. Examples of this type of vocabulary are: schemas (metadata schemas, RDF schemas) and ontologies. (As an example, SKOS is a description vocabulary, an RDF schema – broadly, an ontology -, for describing concepts and concept schemes and their relations.)

2) Value vocabularies or “Knowledge Organization Systems” (KOS) are the vocabularies that define concepts and things that can then be reused in other datasets. Examples of this type of vocabulary are: thesauri, subject headings, taxonomies, classifications (so yes, also statistical classifications are a type of vocabulary!)

In the case of the Caliper Vocbench, the classification that we are creating/importing in the Vocbench is a value vocabulary or KOS, while the models we use (SKOS, XKOS, DCAT) are description / modelling vocabularies designed to describe classifications and datasets. If these modelling vocabularies are not enough, we need to find or create additional ones.

What we need for refining the data model in the Vocbench are RDF schemas or an RDF ontologies.

The classes and properties from the additional schemas or ontologies will be added to the basic SKOS/XKOS/DCAT model in the Vocbench and will provide more metadata elements for the concepts.
Identifying an appropriate schema or ontology

So, before deciding to create a new data model for a new classification, it is recommended to check if a suitable vocabulary already exists.

Unfortunately, there isn’t a global catalog of all existing RDF vocabularies. A good idea is to first ask someone who works with RDF and the Semantic Web. You can also search on Google, for instance “RDF schema for animal species” or “RDF vocabulary for geographic names”.

In addition, there are some good catalogs to search:

- Linked Open Vocabularies (LOV) - https://lov.okfn.org/dataset/lov
- The Basel Register of Thesauri, Ontologies and Classifications (BARTOC) - http://bartoc.org/
- FAIRsharing - https://fairsharing.org/
- GODAN Action Map of data standards - http://vest.agrisemantics.org
- AgroPortal - http://agroportal.lirmm.fr/
- Planteome - http://browser.planteome.org/amigo

If you search on these catalogs, consider that what we need is: a) a schema or an ontology; b) since we’re working in RDF, more precisely an RDF schema or ontology; c) a schema or ontology that describes the entities we have in our classification (for instance, if we need properties for crops, we may want to search for crop ontologies, or if we need properties for geographic entities, we can look for geospatial or geopolitical vocabularies).

Ontologies tend to be very comprehensive and may sometimes be an overkill for modelling a statistical classification, but you can just reuse one or two properties from an ontology.

An example of a classification that may need some additional properties besides the SKOS ones is the Crop List of the World Census of Agriculture (WCA): that list has both vernacular names and scientific names of crops. We can map the vernacular name into the standard skos:prefLabel property. We can also map the scientific name roughly with the skos:altLabel property (an alternative name). But if we want a specific property for the scientific name, we may want to find a vocabulary that contains it. The Darwin Core RDF vocabulary (http://rs.tdwg.org/dwc/terms/) was designed by GBIF to describe species and it has the vernacularName and scientificName properties for the class Taxon. We could decide to import this vocabulary and reuse some of its properties in the Vocbench.

Once you have identified a suitable vocabulary, either download the RDF or copy the vocabulary URI, to then import it into the Vocbench.

Creating a new schema or ontology

There are dedicated tools to build ontologies or RDF schemas. Some are very technical, some are more intuitive. The most used one is Protégé (https://protege.stanford.edu/), with its web-based version Web Protégé (https://webprotege.stanford.edu).

We would recommend to use the Neologism platform, as it is very simple and an instance was deployed in a EU project (agINFRA) in which FAO was a partner, which means we can use it: https://vocabularies.aginfra.eu/.

[Instructions on how to create a schema in Neologism will be provided separately.]

Creating a schema or ontology in the Vocbench

42
[Coming soon]

Importing a schema or ontology in the Vocbench

Whether you have identified an existing suitable ontology or you have created yours, you have to import it in the project where you need it before being able to use it.

NOTE: even if you have created / edited the schema / ontology in the Vocbench, ideally in a dedicated project, you will anyway need to import it in the project where you need it.

Once you have identified a suitable vocabulary, either download the RDF or copy the vocabulary URI, to then import it into the Vocbench project. (If you created it in the Vocbench, go to the project where you created it and download it from the “Export data” link under Global Data Management. Or if you already published it at a public URI, use the public URI.)

Let’s use as an example the XKOS vocabulary, which we have already imported in the existing Caliper Vocbench projects as it is part of the Caliper model. Let’s see how to import it into a new project. If you search Google and go to the XKOS specification, you will see that the URI for this vocabulary is [http://rdf-vocabulary.ddialliance.org/xkos#](http://rdf-vocabulary.ddialliance.org/xkos#).

Let’s import this vocabulary for instance in the CPC 2.1 classification, where we want to state that CPC 2.1 “supersedes” CPC 2.0 (SKOS doesn’t have a property for this).

Go to the project where you need to use the ontology (in this case CPC 2.1) and click on “Namespaces and imports” under Metadata:

![Namespace management page](image)

In the Namespace management page, you will see all the vocabularies (schemas and ontologies) that are already loaded in the Vocbench (this may differ from instance to instance). The “namespaces” represent both conventional machine-readable names that identify the vocabularies and the locations where they are published. They’re associated to prefixes that are used in the interface and in the RDF representation to refer to the vocabularies.

To import a new vocabulary, click on “Add import from” and select one of the following options:

- **Web** (or “Web to mirror”) if the schema is public and you have a public URL from where it can be imported (in many cases the URI is enough, in other cases you need the URL of the actual RDF file).

  “Web to mirror” can be used if you think you will need the same schema in other projects: it will create a mirror of that schema that can be used in all local projects.
- **“Local file”** if the ontology is not published and you have it in a local RDF file. The local file import will always create a mirror available to all projects.

In the case of the XKOS vocabulary, it has been published and its URI is [http://rdf-vocabulary.ddialliance.org/xkos#](http://rdf-vocabulary.ddialliance.org/xkos#). So you can select Web in the “Add import from” dropdown. In the import popup, copy the URI (in most cases, it is good to tick the “Force format” checkbox):

After importing the schema, it is also good to associate its namespace to a prefix, so that the Vocbench interface uses the prefix instead of the whole URI: click on the plus (+) icon on the right of the “Namespace management” box:
And add the prefix / namespace mapping:

Then, you will see both the import and the prefix mapping:
Once a new schema / ontology is imported, under the Class and Property tabs you will find all the classes and properties from the new schema: see for instance all the new properties with the xkos: prefix:

Once the properties are there, you can add them to the description of the scheme or the concept. For instance, go under Scheme, click on the scheme and add a new property to the right panel: since we will have to insert the URI of another scheme, let’s select “Add value manually” in the dropdown of the other properties section:

In the popup, you now have the xkos: properties. We can now use a property to say that this classification supersedes another one.
And copy the URI (the URI of the CPC 2.0 is http://stats-class.fao.uniroma2.it/cpc/cpc_v2.0/), enclosing it between < and > to indicate that it’s a URI:

Now the xkos:supersedes property of the CPC 2.1 is set to the CPC 2.0:

5.2 Importing a classification with a custom data model

This is an advanced scenario and may need the support of a Vocbench expert.

The overall idea is that you identify and import an additional schema or ontology (as we did in the example of importing the XKOS vocabulary) that allows you to represent the additional fields/properties you need, and then when you import the classification you adjust the template to add the new properties, using their RDF notation as we saw in the previous CSV import example.

M49 FAO example

Let’s use an example from a real case.

The M49 classification of geographic areas, in the FAO version which includes corresponding codes in different systems and geographic and economic regions, requires a slightly more complex model than the SKOS one.
NOTE: the model of the M49 as used in FAOSTAT differs from the common code list model under four respects, which could in some way be represented using the SKOS and DCT vocabularies used in the Vocbench:

a) It distinguishes between official names and short names: SKOS would be enough to represent this, if we considered the short names just as alternative labels (skos:altLabel)
b) It indicates validity start date and end date for a geopolitical entity: this could be represented using the dct:valid property.
c) It indicates the type of geopolitical entity (country, region...): this could be represented using the dct:type property, although ideally the types should be referred to through URIs.
d) It includes the corresponding codes used in other geographical code lists, including ISO: this could be represented in SKOS in two ways:
   a. Using the skos:exactMatch property, linking to the same concepts in other geo code lists, but: 1) the other code lists have not been published as Linked Data so there are no URIs to link to; 2) what is useful for FAO statistical applications is to have the codes right in the code list, while with skos:exactMatch a query would have to then get the code from the linked concept.
   b. Using skos:notation and distinguishing the different notations through a DataType (this is the solution recommended in the XKOS specification), but: a) this would require the definition of new custom data types in a local ontology and Vocbench users would need to learn how to use them; b) the Vocbench does not visually represent the differences between notations with different data types, so all notations would be visually indistinguishable.

The model above could be considered as an alternative for representing M49 just as SKOS, so it can be implemented anyway, even if the specific ontology-supported solution below is implemented.

Let’s say that we decided to use an additional ontology that supports all the additional properties we need.

Steps that we need to perform:

1 – Create / open the project where you want to create the M49 FAO classification (see chapter 1.2)
2 – Create the new scheme for M49 FAO (see chapter 1.3)
3 - Identify and import a suitable ontology (chapter 5.1 and more details in 5.2.1 below)
4 - Import the data or “concepts”, in our case geographic areas, mapping them to the properties of the imported ontology (chapter 5.2.2 below)

5.2.1 Identifying and importing a suitable schema / ontology

Let’s remember that

a) RDF data models are not tabular but semantic (relational class-property models, as in UML)
b) RDF data models have to be formalized in a schema or ontology

so we should not expect to just be able to define column names or dimensions in the VB for the columns we want to import. We have to make sure that we have the necessary schema/ontology with its classes
and properties (the semantic relational model) in the VB project to map the column names of the files we want to import.

Looking at the two master files of the M49 FAO (one for the countries and one for the geographic groups), we can see that we need properties that are not in SKOS, like ISO2 code, start year, short name etc.:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Identifier</td>
<td>Code FAO</td>
<td>Code RO3</td>
<td>Code ISO2</td>
<td>Code UNSD</td>
<td>Start Year</td>
<td>End Year</td>
<td>Area Type</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>2</td>
<td>1001</td>
<td>2</td>
<td>AFG</td>
<td>AFG</td>
<td>1900</td>
<td>9999</td>
<td>TRUE</td>
<td>Country</td>
<td>Afghanistan &amp; Tajikistan</td>
<td>Afghanistan</td>
</tr>
<tr>
<td>3</td>
<td>1002</td>
<td>248</td>
<td>ALA</td>
<td>ALX</td>
<td>2003</td>
<td>9999</td>
<td>TRUE</td>
<td>Country</td>
<td>Albania</td>
<td>Albania</td>
</tr>
<tr>
<td>4</td>
<td>1003</td>
<td>3</td>
<td>ALB</td>
<td>ALB</td>
<td>2012</td>
<td>9999</td>
<td>TRUE</td>
<td>Country</td>
<td>Albania</td>
<td>Albania</td>
</tr>
</tbody>
</table>

File 1 – M49 Master file of all countries

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Global Name</td>
<td>Global Code</td>
<td>Region Name</td>
<td>Region Code</td>
</tr>
<tr>
<td>2</td>
<td>World</td>
<td>001</td>
<td>Asia</td>
<td>142</td>
</tr>
<tr>
<td>3</td>
<td>World</td>
<td>001</td>
<td>Europe</td>
<td>150</td>
</tr>
<tr>
<td>4</td>
<td>World</td>
<td>001</td>
<td>Africa</td>
<td>002</td>
</tr>
<tr>
<td>5</td>
<td>World</td>
<td>001</td>
<td>Africa</td>
<td>142</td>
</tr>
</tbody>
</table>

File 2 – FAO M49 master file of geographic groups (4 levels) by country

The second file is all about the hierarchy, and for this we have the necessary properties in SKOS (broader and narrower). We’ll see later how to customize this file to map columns to these properties.

We need to find a vocabulary that has properties for the data in the first file. If you search on Google for “ontology for geopolitical entities”, you will find several links to an ontology that FAO developed some years ago for geopolitical areas: the FAO Geopolitical Ontology. This ontology was designed with the same purpose as the M49 FAO authority file: providing official long names and short names plus the corresponding codes in other systems and the relations between the different types of areas (countries, geographic regions, economic regions). This ontology is not published under a URI (as it hasn’t been maintained in the last years) but we can find the RDF file on the Internet and reuse it. **(We also updated it following the recommendations of the latest consultation held in FAO a few years ago.**)


As we explained in the previous chapter, if an ontology is not published at a public URI, you have to save the RDF file locally and import it into the Vocbench from your computer.

So, open the project created for the M49 FAO scheme, create the scheme (we will call it “M49 for FAO”; and give it a URI, for instance [http://stats-class.fao.uniroma2.it/geo/m49fao/](http://stats-class.fao.uniroma2.it/geo/m49fao/)).

In that project, follow the procedure for importing ontologies that we described in the previous chapter, and this time select “Local file” in the “Add import from” dropdown:
Then in the popup browse your computer to find the local file, give a name to the mirror file and provide the base URI of the ontology. IMPORTANT: even if the ontology is local and hasn’t been published, it has a base URI: the base URI is contained in the RDF file, usually in an element like this:

```xml
<rdf:RDF xml:base="http://stats-class.fao.uniroma2.it/ontologies/geopolitical#"
```

Copy the base URI of the Geopolitical Ontology (http://stats-class.fao.uniroma2.it/ontologies/geopolitical#) to the BaseURI textbox, browse your computer for the local copy and upload it:

Then map a prefix to help you identify the properties of this ontology in the Vocbench: click on the + button under Namespace management:
And add a prefix (like geopol) mapped to the base URI of the Geopolitical Ontology:

Now let’s check the data model to see how it fits our classification.

Under Data > Property you can see a number of properties to which you can map your columns:

The properties for all the types of country codes, country names and dates of validity are there. Now, the data model is ready. You can import the concepts.

5.2.2 Importing the concepts

File 1 - Countries
Let’s start with adapting the first file, the one with the data about countries, without hierarchy, to the template we described under 3.1.

Let’s replace the column names with the corresponding RDF properties, as we did in the import example in 3.1, using skos: and geopol: properties as appropriate. First 8 columns (7, because as in our first example we remove the identifier column):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>004</td>
<td>002</td>
<td>AFG</td>
<td>AF</td>
<td>AFG</td>
<td>1900</td>
</tr>
<tr>
<td>248</td>
<td>248</td>
<td>041</td>
<td>ALA</td>
<td>AX</td>
<td>ALB</td>
<td>2003</td>
</tr>
<tr>
<td>008</td>
<td>008</td>
<td>003</td>
<td>ALB</td>
<td>AL</td>
<td>ALB</td>
<td>1900</td>
</tr>
</tbody>
</table>

Columns 11-23: (we’ll skip columns 9 and 10 - Public and AreaType - for now)

Now the file is almost ready to import but we need to map two other columns which are not immediately obvious: Public and AreaType:

<table>
<thead>
<tr>
<th>Public</th>
<th>AreaType</th>
<th>Name_EN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>Country</td>
<td>Afghanistan</td>
</tr>
<tr>
<td>TRUE</td>
<td>Country</td>
<td>Albania</td>
</tr>
<tr>
<td>TRUE</td>
<td>Country</td>
<td>Antarctica</td>
</tr>
<tr>
<td>TRUE</td>
<td>Country</td>
<td>Antigua and Barbuda</td>
</tr>
<tr>
<td>TRUE</td>
<td>Country</td>
<td>Argentina</td>
</tr>
<tr>
<td>TRUE</td>
<td>Country</td>
<td>Armenia</td>
</tr>
<tr>
<td>FALSE</td>
<td>Area n.e.s.</td>
<td>Areas not elsewhere specified</td>
</tr>
<tr>
<td>FALSE</td>
<td>Area n.e.s.</td>
<td>Areas not elsewhere specified and unknown</td>
</tr>
</tbody>
</table>

There are no properties in SKOS or in the Geopolitical Ontology to which we can map these columns.

However:

1) **Public**

   There is a property in the OWL vocabulary (loaded in the VocBench by default) called “deprecated” which takes boolean values, so we can use that one, but we need to reverse the values in that column, doing a simple replace of TRUE to FALSE and one of FALSE to TRUE: so that column becomes:
2) **Area type**

“Area type” contains values that state of which type these entities are. In RDF, the type of something is normally represented with the rdf:type property and with a URI for the property value (the URI of a Class that represents that type).

If you go under Data > Class, you will see the additional classes contained in the geopol:ontology:

The Geopolitical Ontology defines classes and sub-classes for different types of geographic
areas. A possible mapping from the countries master file is:

- **Country** > geopol:self_governing
- **Area n.e.s.** > geopol:other

When we import the other types of geographic areas later (geographic regions and economic region) we will see that we have the corresponding classes also for those in the geopol: vocabulary.

In the countries master file, we can use rdf:type as column name (the property) and the area type classes as values in the cells: we simply replace:

- “Country” with: 
  “http://stats-class.fao.uniroma2.it/ontologies/geopolitical#:self_governing”
- “N.e.s” with: 
  “http://stats-class.fao.uniroma2.it/ontologies/geopolitical#:other”
- *(We will also replace self_governing with geographical_region for Antarctica?)*

Again, using URIs instead of simple strings makes these types univocal and unambiguous.

And don’t forget to add the column for skos:inScheme, which will have in all the rows the URI of the new M49 concept scheme you created.

Now, with these two last columns mapped, your file is ready to be imported.

Let’s go to Sheet2RDF under Tools (in the VB project you created for M49 FAO):

You will see that the columns in green are those using skos, which is a default schema in the VB. The other columns with geopol: properties are red, because they still need an additional step to make VB aware of them:
If you click on the Play icon to generate the script, you will see that this time you have to adjust the script.

In the script on the right, replace each instance of %pls_provide_a_converter% with:
literal^^xsd:string

and then click on Play in the right panel to check the import:
And finally import the data clicking on the “Add triples” button in the bottom panel:

---

File 2 - Regions and groupings

The second master file contains the regions and the groupings:

![Image](image1.png)

**File 2 – FAO M49 master file of geographic groups (4 levels) by country**

With this file, you need to do two things:

1) import the geographic areas that are not countries (with the first file you only imported countries)

2) import the hierarchical relations

A. Import the geographic areas

For the first task, you need to adapt this file to a template similar to the one you used for countries, leaving empty or removing completely the columns that are not relevant for regions (e.g. ISO codes, short names etc.). YOU SHOULD FILTER ONLY THE REGIONS OF ANY LEVEL IN THIS FILE.

An easy way to do this is to: a) remove the country columns; b) just use the first two columns for all names and codes: move all names and codes of columns 3 and 4 under columns 1 and 2, then all names and codes of columns 5 and 6 again under columns 1 and 2, then also names and codes of columns 7 and 8. After that, remove duplicates. Once you have names and codes, adjust column names (and order optionally) to the template.

Remember to add the columns for rdf:type (this time is the geopoi: class called geographical_region), and for skos:inScheme and skos:topConceptOf (in this case the only top-level concept is World).

*N.B.: if you want to import also the FAOSTAT codes of the regions, which are not in the groupings file (or
other information that is not in this file, like the names in other languages), you need another file of all regions with their codes and other information: you can adjust that file to the template you used for countries, leaving out what you don’t need.

This modified file could look like this:

The rdf:type this time is the geopol: class called geographical_region. Even though the hierarchy files distinguished between global, regions, sub-regions and intermediate regions, they’re all regions (those distinctions only have to do with the hierarchical level which we will import later).

Import the new file through Sheet2rdf: preview it:

Then click again on the Play button and in the script on the right again replace any instance of %pls_provide_a_converter% with: literal^^xsd:string, then click on Play in the right panel and finally on the “Add triples” button in the bottom panel:

Now you have all the geographic areas (countries and regions) in the VB, without any hierarchy. To see them, first make sure to deselect the scheme under Data > Scheme and then go under Data > Concept: you will see the flat list of geographic areas:
A. Import the hierarchic relations

The template for the hierarchic relation is very simple: two columns: skos:Concept and skos:broader. In this classification, each geographic entity (whether country or region) has only one direct parent area, which in turn may belong to a broader area and so on. So the first column will include ALL codes of ALL geographic entities (both countries and regions), each repeated only once. The second column will include the corresponding immediate parent region.

To create such a file starting from the master file of regional groupings:

3) copy the content of the columns Global code and Region Code in the new template, then remove duplicates;
4) then copy the content of the columns Region code and Sub-region code in the same new template, adding below the previous ones, and again remove duplicates;
5) then do the same for the content of columns Sub-region code and Intermediate region code, and again remove duplicates.

6) At this point you’ll have the broader in the first column, so shift the columns and call the first one skos:Concept and the second one skos:broader.

You should get something like this (sorted by the skos:Concept column):

<table>
<thead>
<tr>
<th>skos:Concept</th>
<th>skos:broader</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>001</td>
</tr>
<tr>
<td>004</td>
<td>034</td>
</tr>
<tr>
<td>005</td>
<td>419</td>
</tr>
<tr>
<td>008</td>
<td>039</td>
</tr>
<tr>
<td>009</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>021</td>
</tr>
<tr>
<td>011</td>
<td>202</td>
</tr>
<tr>
<td>012</td>
<td>015</td>
</tr>
<tr>
<td>013</td>
<td>419</td>
</tr>
<tr>
<td>014</td>
<td>202</td>
</tr>
<tr>
<td>015</td>
<td>002</td>
</tr>
<tr>
<td>016</td>
<td>061</td>
</tr>
<tr>
<td>017</td>
<td>202</td>
</tr>
</tbody>
</table>

If you import this file with Sheet2rdf, you will see that VB understands it immediately (green headers):

Click on the Play button in the left panel, then on the Play button in the right panel, then on the “Add triples” button in the bottom panel and the hierarchy will be imported.

Check under Data > Concept:
Clicking on a country, you can see in the right panel the additional properties of the Geopolitical Ontology: look at South Sudan:
6. VERSIONING

Dataset versioning

Different versions of the same classification often have different code notations or different hierarchies: in this case, they should be treated as separate schemes, even in separate projects, and especially concepts should not have the same base URI.

If we are sure that concepts across versions maintain the same exact notation, then it could be useful to put versions in the same project and use the same base URI for all concepts, so that the only difference between versions is whether a concept belongs to one or not (hierarchies could also be different, using a trick that can be explained in a separate document). This approach has been used for the HS classification, whose versions basically differ because some concepts have been added and some removed, but the codes have not been changed or reused.

Once the schemes for the different versions have been created, and have been declared also as dcat:Distribution instances, and the related dcat:Dataset instances have been created, then the versioning relation between datasets can be set using the dct:versionOf property from the new version dataset to the original version dataset.
Other relations can be represented as described in the first chapter. Note that prov: properties (like prov:wasDerivedFrom) can be used only if the dataset has been declared also of type prov:Entity.

**Distribution versioning**

Schemes / distributions can be versioned using the owl:versionInfo property.

Relations can be set also between schemes:
- dct:versionOf
- xkos:follows / xkos:supersedes
- xkos:variant
- if the scheme has been also declared of type prov:Entity, the various prov: relationships can be used.

7. **ROLES AND PERMISSIONS IN THE VOCBENCH**

VB allows to define roles based on the combination of different “capabilities”: capabilities are a combination of which operations (Create, Read, Update, Delete and Validate, CRUDV) can be performed on which parts of the project (the whole project, users’ permissions, all resources including vocabulary, only SKOS resources, all RDF store, SPARQL...).

The roles in the Caliper VB platform have already been set up by administrators (new roles with a new combination of permissions can be discussed with the admin team).

The main roles are:

- Lexicographer: editors who have permissions to add labels and notes, except validating them; permissions can be limited to certain languages
- Mapper: editors who can create alignments with external concepts
- Ontologist: vocabulary experts who can edit classes and properties
- Thesaurus editor: lexicographer permissions + permissions to manage concepts (add, remove, update) and to execute SPARQL queries
- Project manager (managers with full permissions on a project)
- Validator (users who can validate draft edits)
- Publisher
- Administrator
Most of editors will have one or both of the lexicographer and thesaurus-editor roles. There are also access restrictions related to schemes: editors can only edit within their assigned schemes.

As concepts and property values go through the proposal > revision > validation > publication workflow, they are assigned a different status: Draft, Revised, Validated, Published, Proposed deprecated, Deprecated.

The History tab shows all past actions, while the Validation tab shows all changes that have not been validated.

**8. PUBLICATION**

When the classification is ready for publication, the RDF has to be exported and uploaded to the Fuseki triple store. Once the data is there, it is visible in SKOSMOS.

This step is at the moment performed by Tor Vergata.

The data from the Vocbench is also imported into the Caliper website, where an alternative browsing interface is available. This step can be performed by the Drupal administrator and in the future will be automated (the Drupal website will request the data from Fuseki at regular intervals and update the classifications).