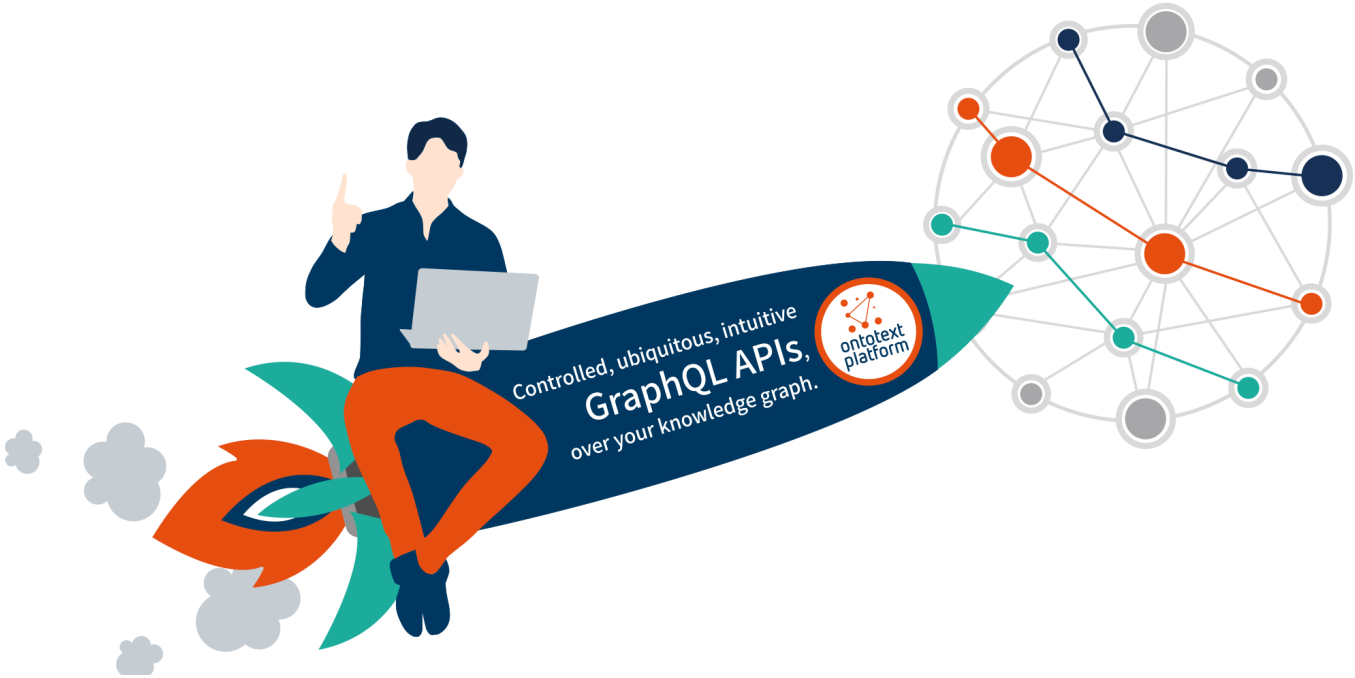
ontotext



Declare, then auto-generate, simple, controlled, intuitive APIs, that are approachable, no matter your background.

Stop wasting time building data access code manually, let the Ontotext Platform auto-generate a fast, flexible, and scalable GraphQL APIs over your RDF knowledge graph.

Let the platform handle the boring grunt work so that you can get on with rocket science.

Fulfil the promise of a controlled, ubiquitous and universal language across all of your data with the platforms optimised GraphQL layer supporting federation and stitching/mesh.

If you are familiar with object-relational/RDF (ORM) mapping, the idea of adding a layer between a database and the applications that use it should sound very familiar to you. Are you an existing GraphDB user who is planning to build an API layer over GraphDB? Perhaps, you have already built one, which is inconsistent, fragile, difficult to integrate with modern front end stacks, does not support the basics such as role-based access control, data shape validation, caching or denial of service limits. Are you having difficulty joining your knowledge graph APIs with other data sources? Maybe you spend an inordinate amount of time and

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

effort managing operational concerns, deployments, monitoring, metrics and log collation?

If so, STOP and give Ontotext platform a try. We have seen these problems many times, and have frankly got tired and bored with the repeating pain, drudgery and waste of vital time and money. We all want to solve the interesting data challenges, build analytics, generate graph embeddings and train smart machine learning models over our knowledge graph data. But well, sometimes we don't get around to the interesting stuff when we waste it needlessly building simple APIs.

Semantic Objects and the soon to be released Semantic Search service (declarative semantic search over Elastic) are different from ORMs in that they are dedicated cloud-native microservices. They allow Ontotext to perform optimizations that are not easy/possible using an ORM embedded within a custom-built API.

The platform includes a sophisticated query/mutation invocation layer that understands the requested objects, properties and relationships. It resolves the request in the most efficient way possible, by producing SPARQL that is optimal for GraphDB's query planner.

If you have ever built your own custom GraphQL API layer, the code typically resolves each part of a GraphQL query as it traverses downwards as a separate isolated data fetching step. This leads to lots of small data fetches to/from GraphDB over the network. As a side effect, the code performs unnecessary data transforms, joins and potentially requires your service architecture to employ cache mitigation strategies to reduce the inevitable latency. Custom code also tends to over-fetch data that is not required.

So, now, you most probably want to avoid trying to develop code to turn complex nested, filtered API requests into the single most efficient SPARQL query? Let Semantic Objects do the hard work of understanding the sometimes bewildering complexity that is SPARQL, by hiding it behind a friendly API so that you don't need to scratch your head.

Custom-built APIs often evolve in an ad hoc fashion, use different protocols (REST, GraphQL, gRPC, etc, etc), use incompatible request and response payloads that don't conform to a single joinable and ubiquitous language. Your application APIs may also be difficult to use with modern application stacks such as React or Angular and you may need to support your own client libraries and tooling.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

So let's be pragmatic, wise and use the Semantic Object workbench's two major workflows to provide a controlled GraphQL standard-compliant API lens over knowledge graph bounded contexts.

Generate a semantic object model from existing RDF schemas

So, you have an existing RDF schema and dataset. Perhaps it's something like the Star Wars schema I discussed in my previous blog posts. Or maybe you are interested in managing a taxonomy and controlled vocabulary of Financial Industry terms? Perhaps you might proceed to design a data model and build an API for this data?

At this point, most RDF folks would re-use the existing, and well used (some might say loved) Simple Knowledge Organisation System (SKOS) ontology to represent a taxonomy. By re-using an existing model effort is minimized, but more importantly, data becomes interoperable as it aligns with a common model.

The EDM Council took this approach with the Financial Industry Business Ontology FIBO. FIBO structures its vocabulary using SKOS. The FIBO vocabulary defines the sets of things that are of interest in financial business applications and represents these things as a tree structure of broader and narrower terms. Starting with the most general, and becoming steadily more specific, (Thing > Autonomous Agent > Organization > Formal Organization > Court of Law), and so on.

Perhaps you now decide to manage the SKOS and FIBO vocabulary in GraphDB, but also want to provide a controlled, consistent, simple to use API across the data. At this point, you could start building one or instead save yourself this pain by generating a GraphQL API with a few UI clicks.

So, let's do this. First of all, we shall put the FIBO & SKOS RDF data into GraphDB. To get up and running, you can follow the platform's installation, and start-up guides, then download the data and subsequently upload the SKOS ontology and GraphDB workbench. Alternatively, if you are a command-line guru, you can make use of the UNIX or Windows scripts that are also described in download (check out README.md for instructions). The scripts will set

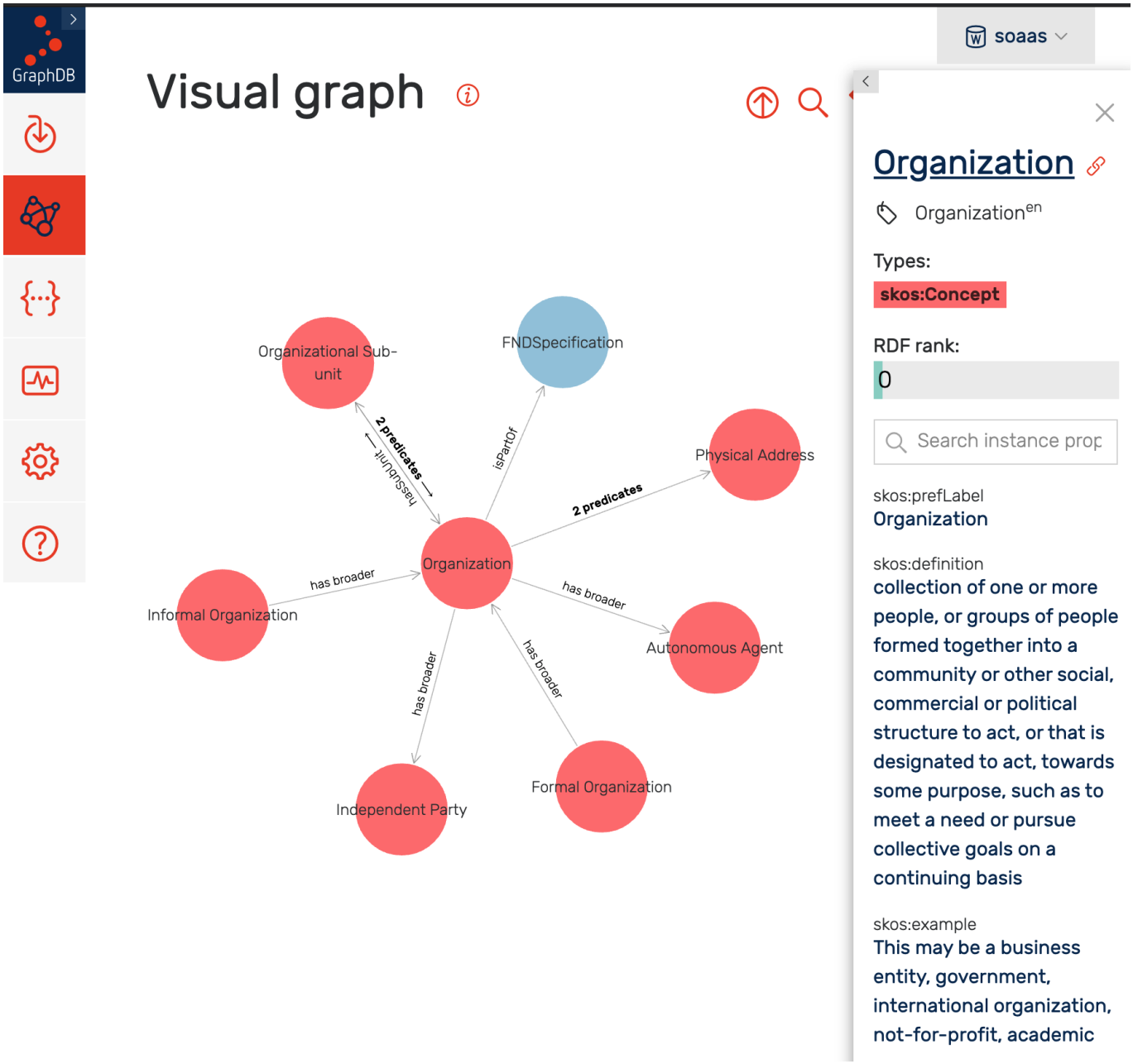
Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

everything up for you, making using the platform APIs, saving you from UI repetitive strain injuries.

In order to start the platform, you will also want to get yourself a platform evaluation license by reaching out to our sales or marketing teams.

So, now you have a graph of data representing the FIBO vocabulary, in a running instance of the platform on your machine. Now, you can go ahead and visualise and traverse to the Organization term by clicking on the following screenshot:

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

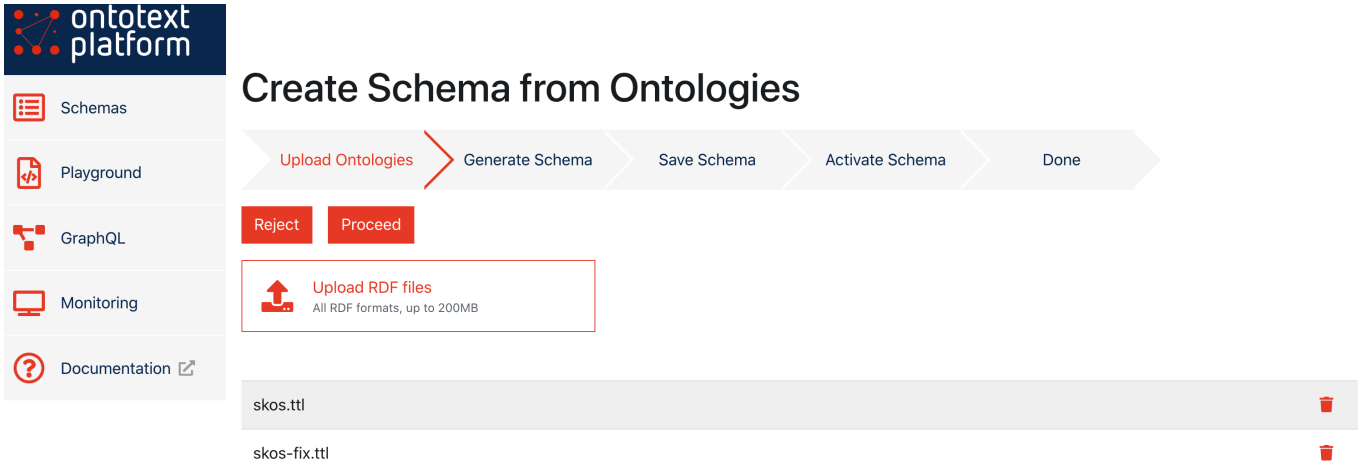


By this point, you might also be interested in exploring FIBO in full using GraphDB, and its inference and property path features if so we totally recommend reading Kevin Tyson's blog post Exploring FIBO Using the Inference and Property Path Features of GraphDB. But for now, we just want to generate a GraphQL API that reflects the SKOS schema and exposes the FIBO

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

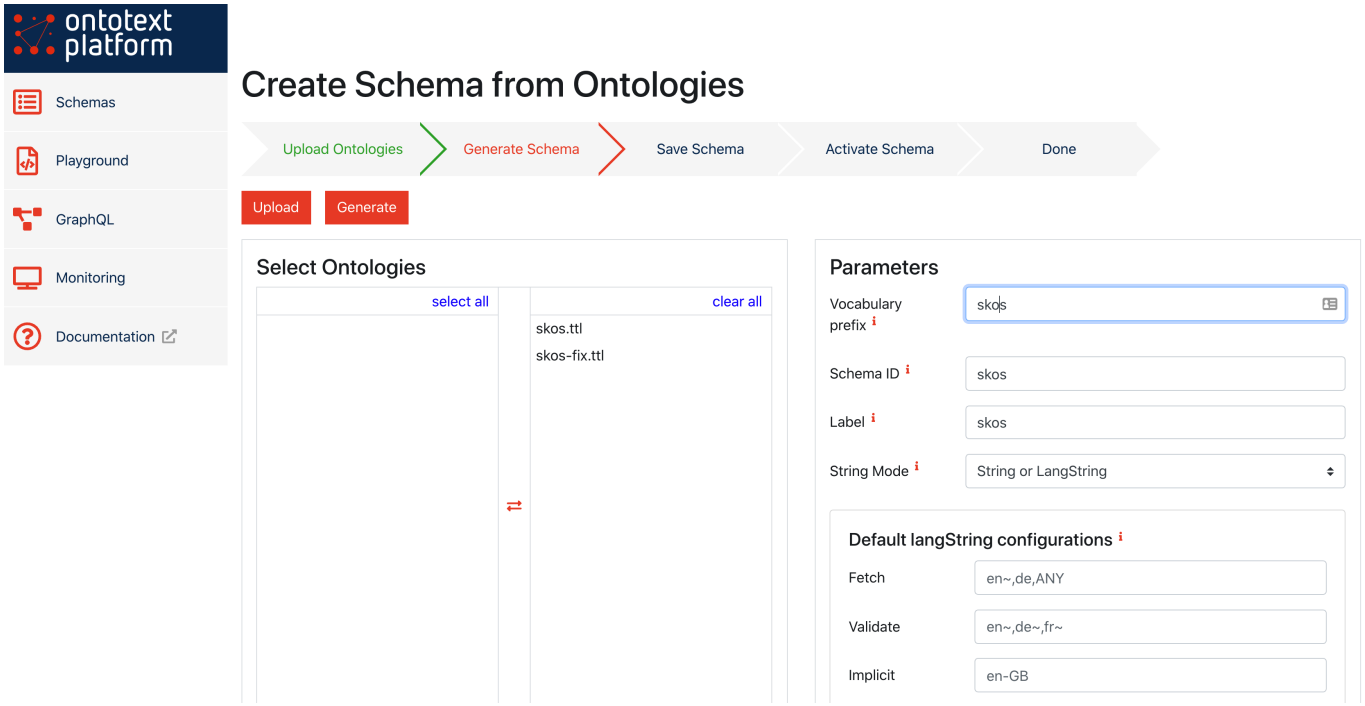
vocabulary.

Let's go ahead and load the SKOS schema within the platform workbench. From the home screen, click **Generate Schema**. This will start the Schema Generation Wizard, after which you can **Upload RDF files** by selecting the skos.ttl and skos-fix.ttl RDF schema files that are included in the download, unzipped previously. The download expands the default SKOS ontology to include a small number of explicit domain/range declarations to overcome SOML's missing property inheritance (which will be included in a future release).



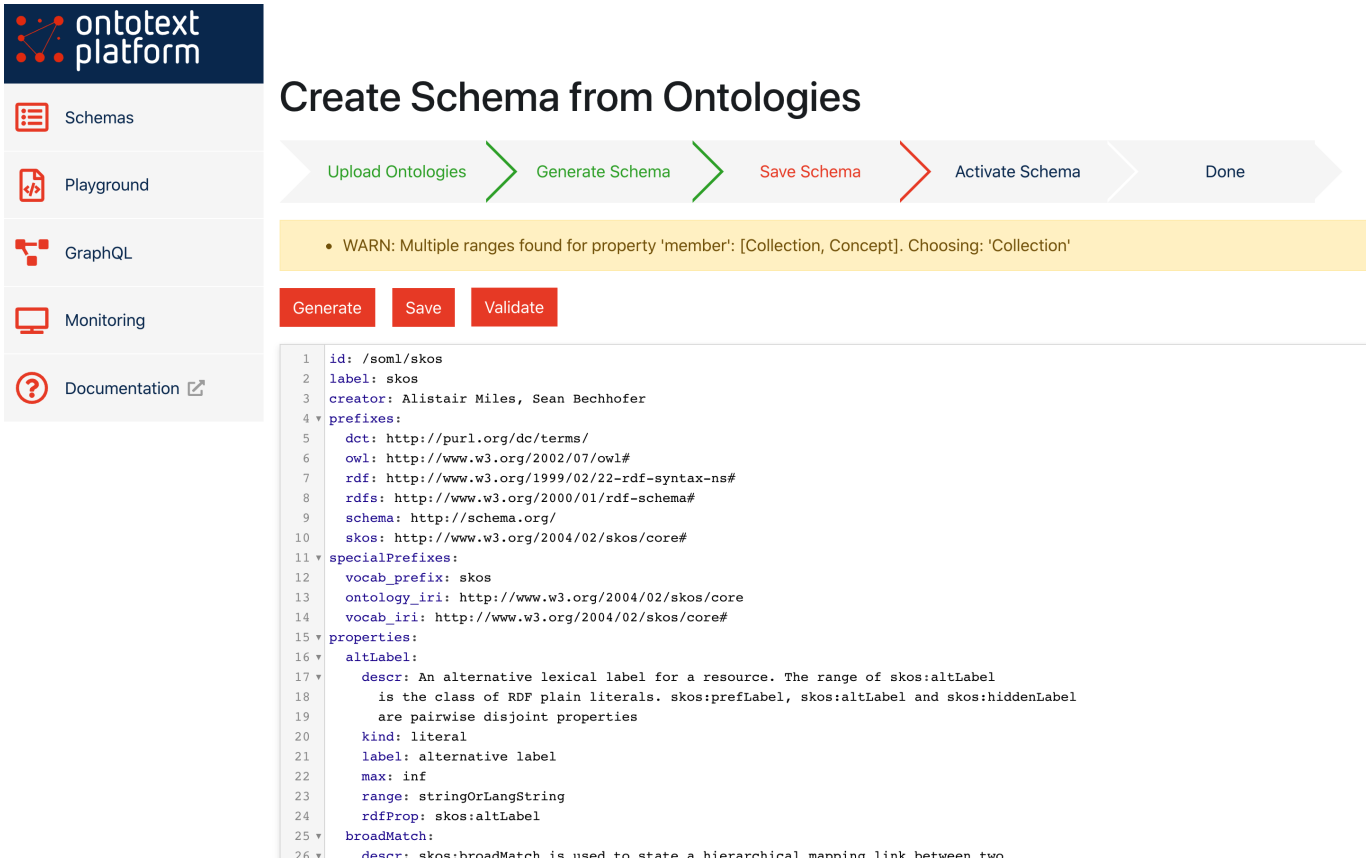
Then it's time to **Proceed** and transform the RDF schema into a Semantic Objects Modelling Language (SOML) schema. But before kicking off the transformation, you will need to give the SOML schema a prefix and an identifier (you can choose "skos"). You can also leave the rest of the defaults such as "String Mode" as is.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com



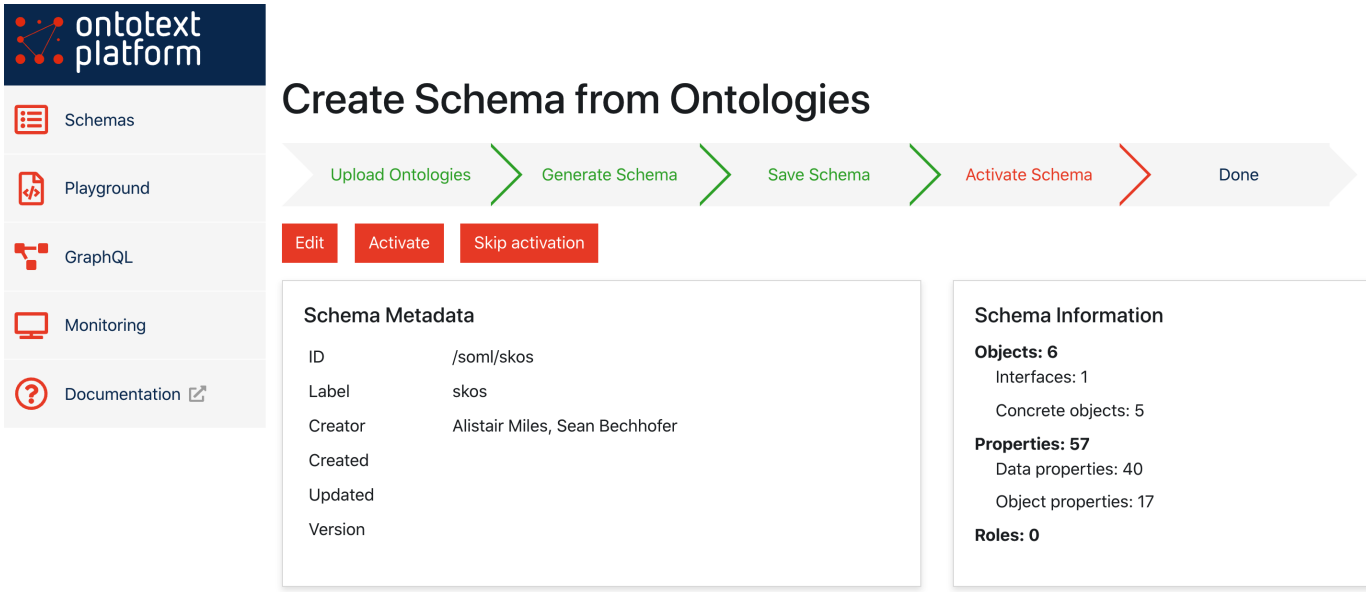
So, now let's **Generate** the SOML and **Save** it (we don't expect to make any changes at this stage). But, if you are feeling adventurous, you can make changes to augment the auto-generated schema with additional features. Perhaps you might want to add RBAC controls or remove some of the Objects or properties so that they don't expand unnecessarily within the bounded context generated API.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

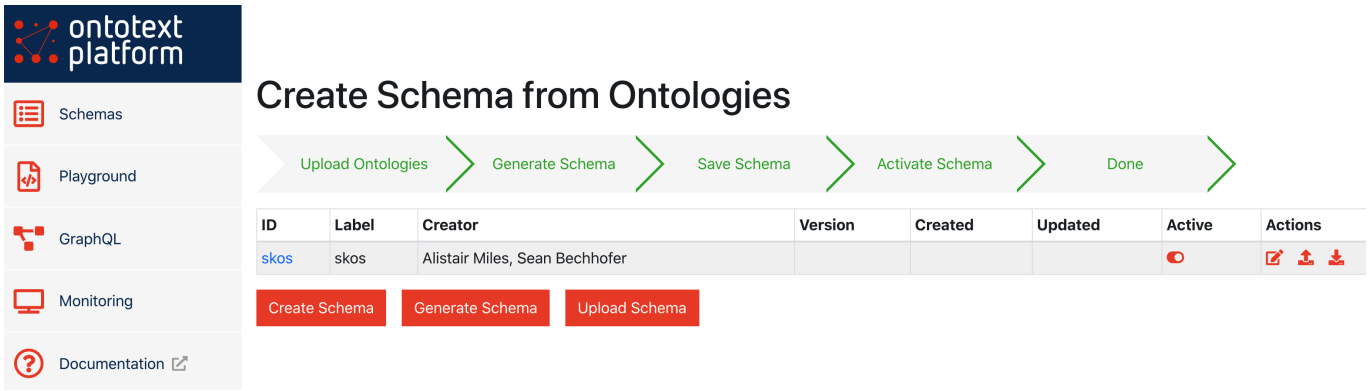


The workbench wizard has so far transformed the SKOS RDF schema into a valid YAML based SOML schema, given it a unique identifier and saved it. The transformation process has also detected 6 Semantic Objects (Collection, CollectionInterface, Concept, ConceptScheme, OrderedCollection, rdf:List) and the 57 related object properties such as (narrower, broader, hasTopConcept, etc).

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com



The SOML schema is also now managed inside the platform's schema library, allowing it to be used and set as **Active**.



Once you have an **Active** SOML schema, the platform binds it to a running Semantic Object service instance. The activation process transforms the SOML schema into a GraphQL schema and auto-generates a controlled GraphQL API at the Semantic Objects GraphQL endpoint. The platform allows you to store as many schemas as you like and bind any one of these schemas to a Semantic Objects service. Thus allowing you to provide a set of controlled lenses over your knowledge graph.

Now you can start using the SKOS GraphQL API. Perhaps you might want to run a query to retrieve the broader and narrower terms for the FIBO organisation term, as follows:

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

```
query {
  concept(where: {prefLabel: {value: {EQ: "Organization"}}}) {
    id
    prefLabel {
      value
    }
    definition {
      value
    }
    example {
      value
    }
    broader {
      id
      prefLabel {
        value
      }
    }
    narrower {
      id
      prefLabel {
        value
      }
    }
  }
}
```

You can go ahead and try this query on the platform workbench you have deployed locally at http://localhost:9993/graphql.

Be sure to check out the documentation and play with all the query and mutation features.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

Manually author a semantic object model

Perhaps you cannot find a suitable existing RDF schema to bootstrap your bound context API? Or perhaps, you have an existing ontology but its depth and breadth are too large for your use-case and bounded context API needs? Or maybe you want to use a set of existing RDF schemas and augment these with your own custom objects and properties?

For instance, you might want to build an API to manage a small controlled lens over DBPedia. For example, you may want a very simple RBAC API to allow users with specific roles to manage **dbo:Athlete** Classes and their **dbo:birthPlace** property, but nothing else across DBPedia's 685 hierarchical classes described by its 2,795 different properties. In this case, it's probably not worth autogenerating a SOML schema from the DBPedia schema as it will be extremely noisy, large and contain lots of objects and properties that you will just want to delete.

Or maybe you want to expand the simple SKOS GraphQL API, which was autogenerated with a couple of additional FIBO specific objects/classes and properties?

So, let's take the **fibo-v-fnd:Organization** FIBO entity as an example and add some additional Semantic Object properties so that the generated GraphQL API is able to manage these data points.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

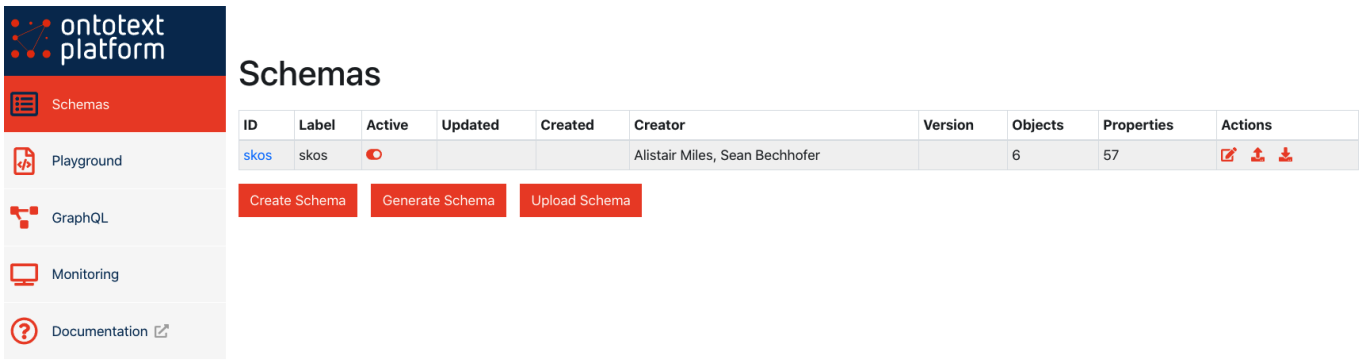
```
@prefix rdf:             <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix skos:            <http://www.w3.org/2004/02/skos/core#> .
@prefix fibo-v-fnd:      <https://spec.edmcouncil.org/fibo/FND/vocabulary#> .
@prefix fibo-v-be:       <https://spec.edmcouncil.org/fibo/BE/vocabulary#> .
@prefix fibo-vocabulary: <https://spec.edmcouncil.org/fibo/vocabulary#> .
```

```
fibo-v-fnd:Organization
    rdf:type                      skos:Concept ;
    skos:prefLabel                "Organization"@en ;
    skos:definition               """
                  collection of one or more people, or groups of people formed
                  together into a community or other social, commercial or political
                  structure to act, or that is designated to act, towards some purpose,
                  such as to meet a need or pursue collective goals on a continuing basis" ;
          """ ;
    fibo-v-be:hasOperatingAddress fibo-v-fnd:PhysicalAddress ;
    fibo-v-be:hasSubUnit          fibo-v-be:OrganizationalSubUnit ;
    fibo-v-fnd:hasAddress         fibo-v-fnd:PhysicalAddress ;
    fibo-vocabulary:hasDomain     "Foundations" ;
    fibo-vocabulary:hasSubDomain  "Organizations" .
```

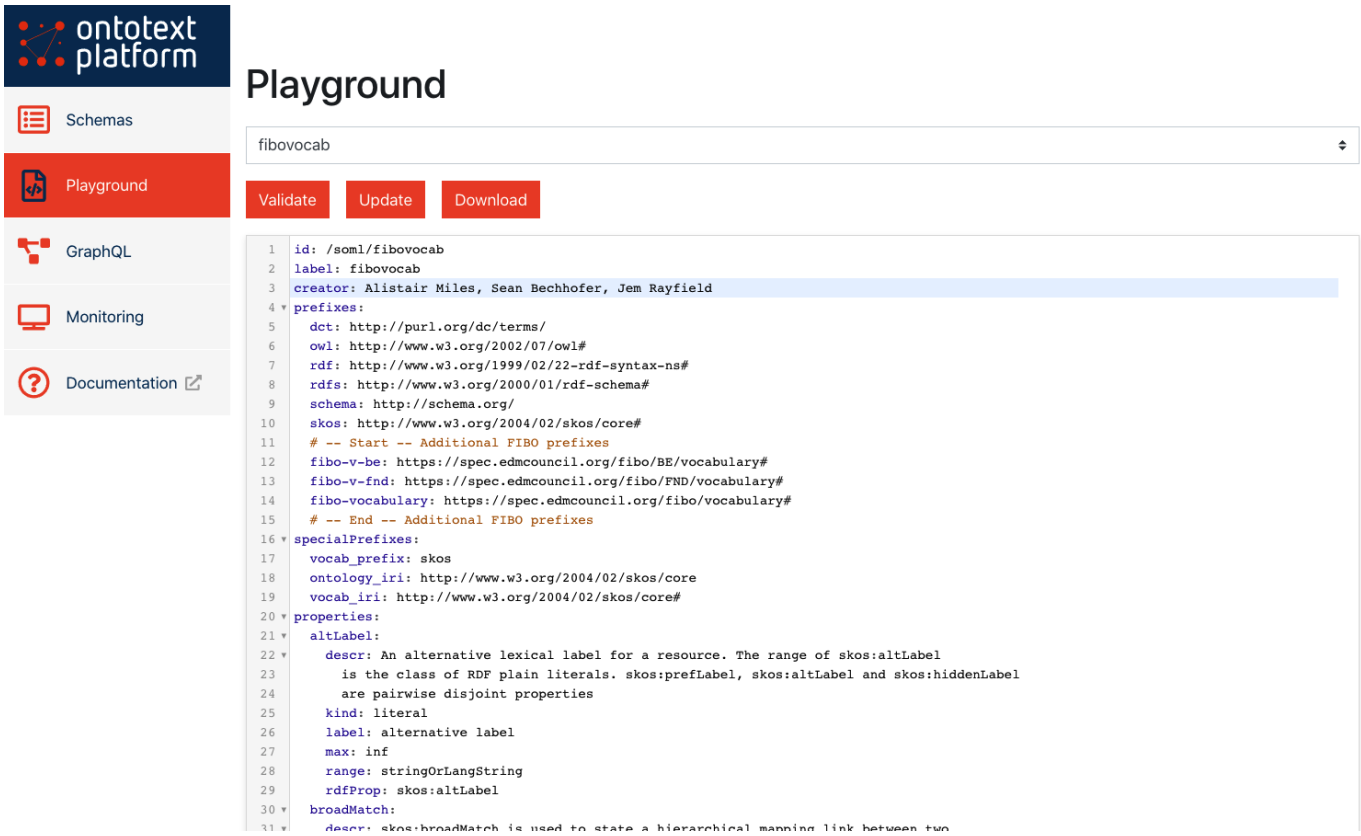
As you can see above, the FIBO vocabulary mixes up attributes of **skos**:**Concept** (prefLabel, definition, hasDomain, hasSubDomain) with properties and ranges of the class corresponding to the concept (eg **fibo-v-be:hasOperatingAddress**, **fibo-v-be:hasSubUnit**). These properties are sub-properties of skos:related. I consider this a non-standard way of capturing an ontology. However, it is simple to reflect this modelling approach within a SOML schema. The following example augments the SKOS schema with these properties, in order that you generate a FIBO vocabulary GraphQL API.

First, things first, let's browse for the existing SKOS SOML schema using the workbench schema library.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com



Then **open** up the SKOS schema into the workbench playground. Copy it and then head back to the schema library so that you can **create** a new schema derived from the autogenerated SKOS schema.



You will then need to add a couple of additional FIBO prefixes in order that Semantic Objects

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

can easily resolve them.

```
prefixes:
  ...
  fibo-v-be: https://spec.edmcouncil.org/fibo/BE/vocabulary#
  fibo-v-fnd: https://spec.edmcouncil.org/fibo/FND/vocabulary#
  fibo-vocabulary: https://spec.edmcouncil.org/fibo/vocabulary#
```

Next, you can add additional FIBO properties. For example, we can declare a **hasOperatingAddress** property as having zero to one (max: 1) SKOS Concepts for its target/range.

```
hasOperatingAddress:
  descr: """
         indicates an address at which an organization carries out operations
         """
  kind: object
  label: has operating address
  max: 1
  range: Concept
  rdfProp: fibo-v-be:hasOperatingAddress
```

And then add the new FIBO properties to the existing Concept object to make them queryable:

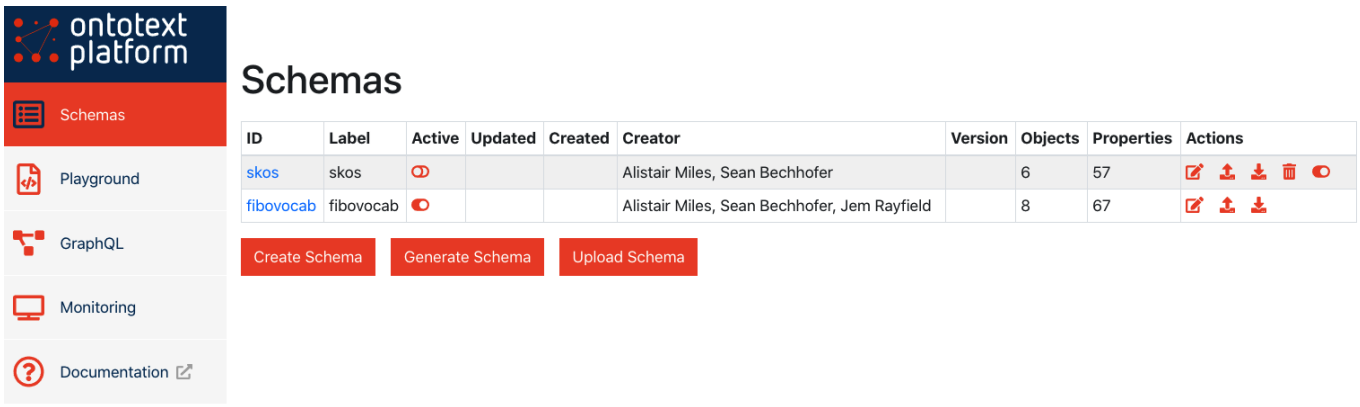
```
Concept:
  descr: An idea or notion; a unit of thought
  props:
    ...
    hasOperatingAddress: {}
    hasSubUnit: {}
    hasOperatingAddress: {}
    hasDomain: {}
    hasSubDomain: {}
    type: skos:Concept
```

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

This can all be achieved by declaring the additional prefixes, properties and object changes using the simple YAML SOML within the workbench playground.

The workbench playground IDE parses, **validate**s and checks, schemas before they are saved and added to the schema library. Thus ensuring that your modifications make sense and can be used to generate a GraphQL API. You can then take the final step and make the schema **active**, so that you may query the FIBO properties as well as the SKOS properties.



For example, the following GraphQL Query retrieves SKOS concept properties and the additional extended FIBO properties hasDomain, hasSubDomain, hasOperatingAddress, hasSubUnit.

```
{
 concept(where: {prefLabel: {value: {EQ: "Organization"}}}) {
  id
  hasDomain
  hasSubDomain
  hasOperatingAddress {
   id
   prefLabel {
    value
   }
  }
  hasSubUnit {
```

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

```
      id
      prefLabel {
        value
      }
    }
    prefLabel {
      value
    }
    definition {
      value
    }
    example {
      value
    }
    broader {
      id
      prefLabel {
        value
      }
    }
    narrower {
      id
      prefLabel {
        value
      }
    }
  }
}
```

You can go ahead and try this query on the platform workbench you have deployed locally at http://localhost:9993/graphql

If you are feeling lazy, the fibovocab schema is included in the download as skosfibo-schema.yaml. By comparison (to skos-schema.yaml), you can discover the small number of manual additions introduced to support the FIBO properties/predicates that extend SKOS.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

Notice in the following GraphQL query, GraphDB's sub-property and symmetric reasoning materialises inferred skos:related properties. It is, therefore, possible to ask "What is skos:related to Organization"?

```
{
  concept(where: {prefLabel: {value: {EQ: "Organization"}}}) {
    id
    prefLabel {
      value
    }
    related {
      id
      prefLabel {
        value
      }
    }
    hasOperatingAddress {
      id
      prefLabel {
        value
      }
    }
    hasSubUnit {
      id
      prefLabel {
        value
      }
    }
  }
}
```

You can go ahead and try this query on the platform workbench you have deployed locally at http://localhost:9993/graphql

The query above includes the explicit FIBO specific sub-properties **fibo-v-**

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

be:hasOperatingAddress and **fibo-v-be:hasSubUnit** in order that you can understand how skos:related is derived and inferred.

Spend your time wisely

Make sure, you keep your eyes peeled on the upcoming Ontotext Platform releases, the platform workbench plans include a Graphical user interface supporting drag-and-drop schema building. Perhaps then, if you are brave enough, you can let your product managers and business analysts loose building APIs so that you can get on with the interesting work.

But there is more, the platform will soon provide declarative GraphQL APIs over ElasticSearch. The Semantic Objects schema will include some additional annotations for configuring analysers, tokenisers, etc., so your RDF data is indexed for smart Semantic Search. Again, saving you time, so that you don't waste it building consistent, joinable search APIs across your knowledge graph.

We want you to avoid developing code to turn complex nested, filtered API requests into the single most efficient SPARQL or Elastic search query!

Ontotext Platform will do the hard work, you can declare a bounded context API, using the platform's workbench, and hide your RDF behind a friendly API so that you don't need to scratch your head.

Then use the platform's text analytics services, including named entity recognition, relationship extraction, and then build your "rocket science". Perhaps, build a classification and recommendation system using the FIBO taxonomy/vocabulary GraphQL API, or let Ontotext's professional services get you there.

Don't waste time on the basics, spend your time wisely and work on your interesting data challenges!

Request Ontotext Platform

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com