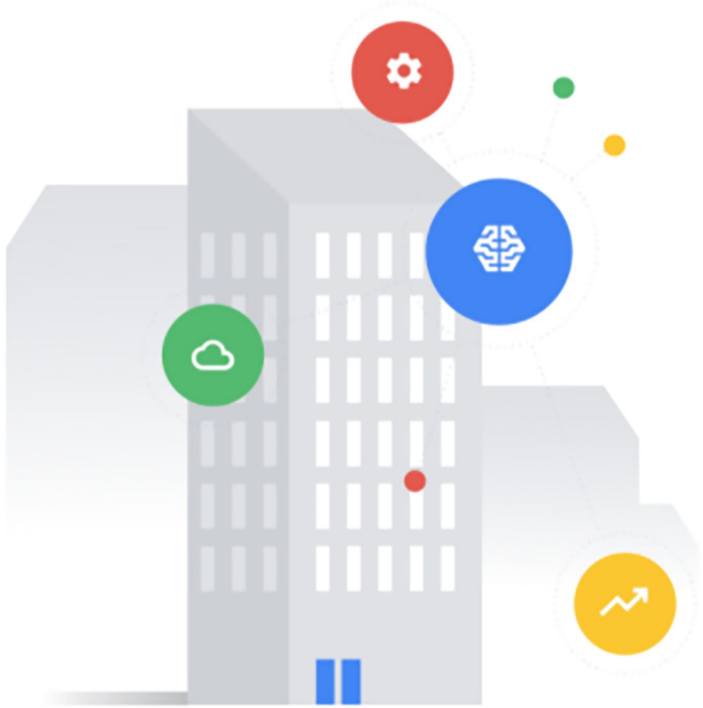
Google

Module 2 | **Lesson 5**

Master Systems Integrator (MSI) onboarding



Before you get started

This onboarding deck has interactive features and activities that enable a self-guided learning experience. To help you get started, here are two tips for viewing and navigating through the deck.

1 View this deck in presentation mode.

- To enter presentation mode, you can either:
  - Click the **Present** or **Slideshow** button in the top-right corner of this page.
  - Press **Ctrl+F5** (Windows), **Cmd+Enter** (macOS), or **Ctrl+Search+5** (Chrome OS) on your keyboard.
- To exit presentation mode, press the **Esc** key on your keyboard.

2 Navigate by clicking the buttons and links.

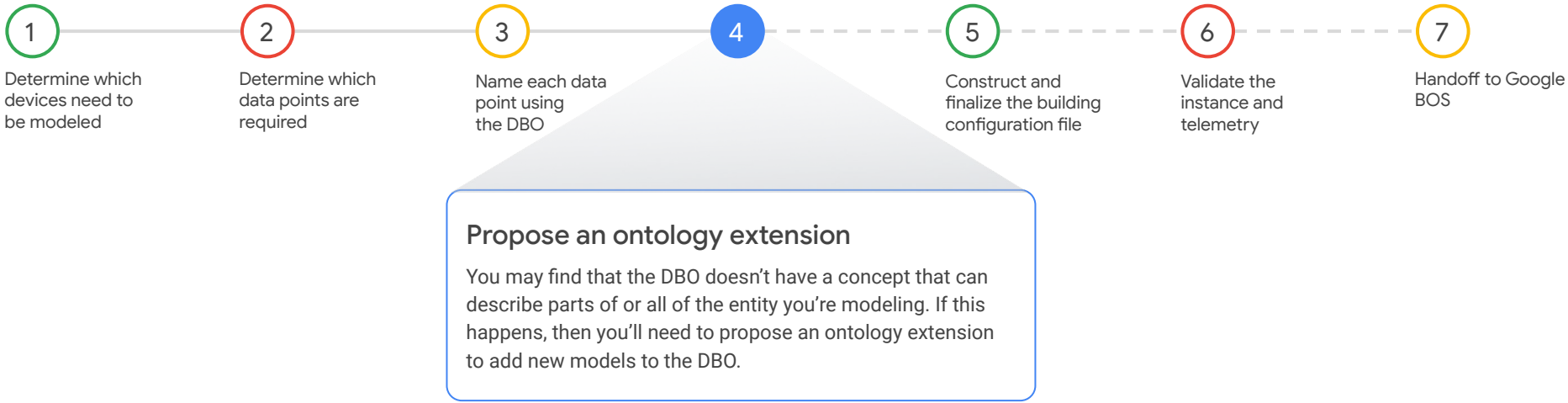
- Click the **Back** or **Next** buttons to go backward or forward in the deck. Moving forward, you'll find them in the bottom corners of every slide.
- Click blue text to go to another slide in this deck or open a new page in your browser.
- For the best learning experience, using your keyboard or mouse wheel to navigate is discouraged.

Ready to get started? Let's go!

Workflow revisited

Here's the recommended workflow for data modeling from Lesson 1.

In this lesson, you'll walk through the fourth step of data modeling with the DBO.



Back

Next

Google

Lesson 5

Propose an ontology extension

What you'll learn about:

- Ontology extensions

By the end of this lesson, you'll be able to:

- Submit a pull request to propose an ontology extension for:
  - Subfields
  - Fields
  - Abstract types
  - Non-abstract types

Back

Next

Google

What is an ontology extension?

While the Digital Building Ontology (DBO) covers a wide array of devices and functionalities, it is impossible to anticipate the needs of every building.

What if the DBO doesn't have what I'm looking for?

If there is a gap in the ontology, you'll need to create the missing item – if it is really needed. You'll then submit a proposal that the ontology be extended to include your new item. The process of creating, submitting, and approving new items for the ontology is called an **ontology extension**.

While any part of the ontology can be extended, the following are extended most frequently:

- Subfields
- Fields
- Abstract types
- Canonical types

For the purpose of this lesson, we will focus on these types of extensions.

Back

Next

Tips for proposing an ontology extension

Here are two general tips to make sure your proposal encounters minimal problems.

1

Double-check that what you need ISN'T already in the ontology.

Have you made sure that what you need isn't already in the ontology, but phrased differently than you expected? Have you checked that what you need isn't described in a sub- or super-set of subfields?

Using the Ontology Explorer, you can see what fields are typically used by common canonical entities. You can note from this exploration what fields are typically used. For example, you might have a **`VAV`** with a damper and airflow control. By using the Ontology Explorer, you can see that a common **`VAV`** (perhaps you choose **`VAV_SD_DSP`** as an example) will have the following required fields:

- `supply_air_damper_percentage_command`
- `supply_air_flowrate_sensor`
- `supply_air_flowrate_setpoint`
- `zone_air_cooling_temperature_setpoint`
- `zone_air_heating_temperature_setpoint`
- `zone_air_temperature_sensor`

From this, you can make some inferences about what fields are typically used for these devices and can then determine if what you are looking for is already modeled somewhere.

```
How would you like to query DBO
1: Get fields for a type name
2: Get types for a list of fields
3: Validate a field name
q: quit

Please select an option: 1
Enter a namespace: HVAC
Enter a type name defined in HVAC: VAV_SD_DSP

Fields for HVAC/VAV_SD_DSP:
/cooling_thermal_power_capacity_: optional
/discharge_air_temperature_sensor_: optional
/manufacturer_label_: optional
/model_label_: optional
/run_command_: optional
/supply_air_cooling_flowrate_capacity_: optional
/supply_air_damper_percentage_command_: required
/supply_air_flowrate_sensor_: required
/supply_air_flowrate_setpoint_: required
/supply_air_heating_flowrate_capacity_: optional
/supply_air_temperature_sensor_: optional
/supply_air_ventilation_flowrate_requirement_: optional
/zone_air_cooling_temperature_setpoint_: required
/zone_air_heating_temperature_setpoint_: required
/zone_air_relative_humidity_sensor_: optional
/zone_air_temperature_sensor_: required
/zone_use_label_: optional
```

Back

Note: For more about using the Ontology Explorer, revisit the walk through of each of its options in Lesson 3.

Next

Tips for proposing an ontology extension (continued)

Here are two general tips to make sure your proposal encounters minimal problems.

2

If you need to propose a lot of changes, submit multiple proposals instead of one large one.

Start by proposing any new subfields, followed by new fields, then new abstract types, and finally any new non-abstract types. By proposing changes in this order, you can correct any issues in lower hierarchy items before those mistakes are repeated elsewhere. For example, if a new subfield has an issue, you won't have to correct that issue in every new field that uses that subfield.

Another effective alternative is to propose your new fields and subfields in a spreadsheet and request a review of this information. Sometimes the context of the collection of fields by entity can make it easier to assess the extensions.

Now let's turn our attention to how to propose different types of ontology extensions.

Back

Next

Google

Creating a subfield

Here the steps for creating a new subfield.

1. Make sure a comparable subfield doesn't already exist.

   There are many subfields already defined in the ontology; you must make sure that what you're creating doesn't already exist. This will require you to use your judgment. Anything you create will need to be justified based on your use case. Reusing an existing subfield might be suggested if what you propose seems superfluous, so be mindful of this as you attempt to create new ones.

1. Determine the proper category for your subfield.
2. Name the subfield and provide a concise description for it.

   For the name, avoid using uncommon acronyms or terminology that isn't intuitive. Try to use terminology already used in the ontology whenever possible.

1. Create a new Git branch and give the branch a descriptive name.
2. Update the subfields.yaml file with your new subfield.
3. Submit a pull request.

   We'll cover this step in more detail in just a bit.

Back

Note: For more on subfields, check out Module 1, Lesson 3.

Next

Google

Creating a subfield

Here are the steps for creating a subfield outlined in an example.

A few things to note about this example:

- The example is run from a machine which has cloned the Digital Buildings Ontology repo from GitHub.
- We will be making ontology updates to the GitHub project, and this requires you to interface with it. The example assumes you're familiar with the process for making GitHub contributions.

Example

	A	B	C	D	E	F
1	**Equpiment Name**	**Point Name**	**Units**	**Description**	**Entity Type**	**Field**
2	EF-1	radon_lvl	PPM	Detected radon level.		
3	EF-1	radon_lvl_stpt	PPM	Radon level setpoint; threshold where the fan turns on and off.		
4	EF-1	fan_ss	NO-UNITS	Fan command to run		
5	EF-1	fan_sts	NO-UNITS	Fan feedback, indicating it is running.		
6	EF-1	fan_alarm	NO-UNITS	Fan alarm, indicating it has failed.		
7						
8						

Let's say there's an exhaust fan for a highly-specialized laboratory space. It's depicted in the BMS points list shown here.

The fan will run based on the detected presence of **radon** gas. It will run when the detected level of radon is above a certain level (say 5 ppm).

Back

*Click **Next** to continue the example.*

Next

Google

Creating a subfield (continued)

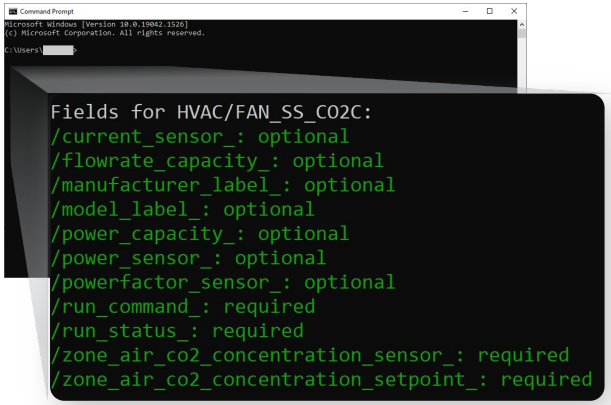
Here are the steps for creating a subfield outlined in an example.

Example (continued)

An inspection of the **`HVAC/FAN`** types show some similar fans, such as **`FAN_SS_CO2C`**, which controls CO_2 gas levels, but no such device for dealing with radon.

```
FAN_SS_CO2C:
  guid: "91bc028c-acbe-11ed-afa1-0242ac120002"
  description: "CO2 control fan."
  is_canonical: true
  implements:
  - FAN
  - SS
  - CO2C
```

And running Ontology Explorer shows the fields for this entity type:



Upon inspection of the ontology, we notice a few things:

1. The subfield "radon" is not defined. It will need to be if we intend to build a specific field for it.
2. The fields **`zone_air_co2_concentration_sensor`** and **`zone_air_co2_concentration_setpoint`** can form the basis for the field we want to create, if we replace "co2" with "radon".
3. Since this will become a canonical type, we must define the subfield, the fields which use that subfield, the abstract type that uses those fields, and the canonical type that uses that abstract type.

Back

*Click **Next** to continue the example.*

Next

Google

Creating a subfield (continued)

Here are the steps for creating a subfield outlined in an example.

Example (continued)

For simplicity, we will create a pull request (PR) that defines all of these things together. If there are a lot of concepts to add, we would do that in several separate PRs, one for subfields and fields, one for abstract types, and one for canonical types. It can be done all at once, but the review process will be much longer.

Here are the updates:

```
protection: "Act of preventing damage to object."
radon: "A radioactive gas; chemical element 86. A noble gas (meaning it is generally non-reactive)."
rain: "Liquid water in the form of droplets that have condensed from atmospheric water vapor."
recovery: "Component or process used for the reclamation of heat."
red: "Red light fracture of ambient light"
```

Some notes:

1. We add “radon” as a descriptor, similar to “co2,” “co,” and “so2.” In this way we simply follow the precedent already set for gas concepts.
2. We add it in alphabetical order (between “protection” and “rain”).
3. We add a descriptive definition. Make sure the meaning is clear, and that its distinction from other terminology already defined is apparent.

Back

*Click **Next** to continue the example.*

Next

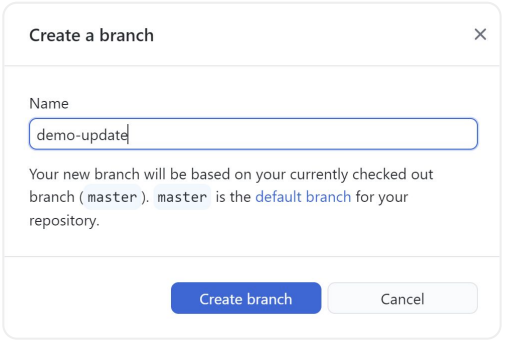
Creating a subfield (continued)

Here are the steps for creating a subfield outlined in an example.

Example (continued)

We have modified our local branch and can create a pull request as a result.

Using GitHub desktop, we can create the new branch in the user interface:



And we can now see the changes that we made in a PR.

Let's continue with the field definitions.

Back

Note: For more on subfields, check out Module 1, Lesson 3.

Next

Creating a field

Here are the steps for creating a new field.

1. Make sure a comparable field doesn't already exist.

   Be sure the field doesn't already exist in a sub- or super-set of subfields.

1. Carefully consider the subfields you use.

   For pointers on selecting the best subfields for your circumstances, check out model_hvac in the GitHub repo.

1. Ensure your proposed field is built with the correct grammar.

   When possible, base your proposed field on a similar field that has already been approved to help create consistency.

1. Create a new Git branch and give the branch a descriptive name.
2. Update the proper .yaml file with your new field.

   Depending on your field, choose either metadata_fields.yaml or telemetry_fields.yaml files.

1. Submit a pull request.

   We'll cover this step in more detail in just a bit.

1. Check the validator results to make sure that your field passes.

Note: For more on fields and the grammar used to construct them, check out Module 1, Lesson 4.

Back

Next

Google

Creating a numeric field

Here are the steps for creating a field outlined in an example.

Example

Having defined the subfield **`radon`**, we can now create fields which utilize it.

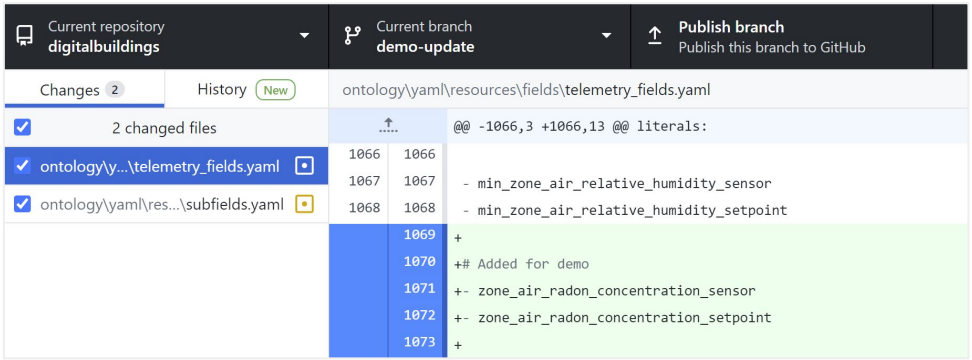
Using what we explored earlier, our new fields must conform to the correct field name syntax. We'll model them after these, which are already curated in the DBO:

- zone_air_co2_concentration_sensor
- zone_air_co2_concentration_setpoint

Here are our new fields, which we'll add to the telemetry_fields.yaml file:

```
- zone_air_radon_concentration_sensor
- zone_air_radon_concentration_setpoint
```

Checking our Github desktop instance, we can see the changes reflected there as well, along with our initial changes.



Back

Note: For more on fields, and the grammar used to construct them, check out Module 1, Lesson 4.

Next

Creating a numeric field (continued)

Here are the steps for creating a numeric field outlined in an example.

For numeric fields, users must define the default value range, i.e., the minimum and maximum expected values for the field across all entities that have the field. The range should be specified as a map containing exactly two entries:

- the minimum, with a key of either **`flexible_min`** or **`fixed_min`**
- the maximum, with a key of either **`flexible_max`** or **`fixed_max`**

Each key should map to a double value, expressed in the SI unit for the field (e.g., Kelvins for **`temperature`** fields).

Fixed means the range should never be changed and should always apply across all entities that have the field (see the second example).

Flexible means the range may change over time or based on the entity. This allows the range to be adjusted by the range calculation pipeline, which periodically calculates new ranges for fields by using the interquartile-range method on historical timeseries data.

A range may consist of two flexible bounds, two fixed bounds, or one flexible bound and one fixed bound.

Example

The following range values are expressed in the standard (SI) unit for **`concentration`**, parts per unit:

```
- zone_air_radon_concentration_sensor
    flexible_min: 0.00005
    flexible_max: 0.005
- zone_air_radon_concentration_setpoint
    flexible_min: 0.00005
    flexible_max: 0.005
```

Fixed ranges often apply to **`percentage`** fields, which may have a range of 0 to 100:

```
- heating_air_damper_percentage_command:
    fixed_min: 0.0
    fixed_max: 100.0
```

Back

Note: For more on fields, and the grammar used to construct them, check out Module 1, Lesson 4.

Next

Creating a multi-state or string field

Here are the steps for creating a multi-state or string field.

Multi-state field

When adding a new multi-state field, first add any new states to states.yaml. Multi-states should contain one of the following subfields: **`alarm`**, **`mode`**, **`status`**, or **`command`**. In the fields .yaml file, the valid states for the field should be listed below the field (visit Module 1, Lesson 5 for more examples).

Example

```
- supply_water_valve_command:
  - OPEN
  - CLOSED
```

String field

When adding a new string field, just list the field by itself.

Example

```
- manufacturer_label
- model_label
- zone_use_label
```

Next, we'll create the abstract type that uses the numeric fields we created.

Back

Note: For more on fields, and the grammar used to construct them, check out Module 1, Lesson 5.

Next

Creating an abstract type

Here are the steps for creating a new abstract type when new functionality is required.

1. Make sure a comparable type doesn't already exist.
2. Carefully build your type.

   Focus on making the type as specific as possible. Make sure your type captures only the distinct point of functionality you're trying to capture. Abstract types are not merely collections of random fields—the fields should have a specific meaning when combined together.

1. Ensure that your type includes the following information:

   - A name that follows conventions.
   - A description.
   - A flag that it is abstract.
   - A list of fields that the type uses, both required and optional.
   - A list of any inherited types it may use.

1. Create a new Git branch and give the branch a descriptive name.
2. Go to the correct namespace for your new type and update the proper .yaml file.
3. Submit a pull request.

   We'll cover this step in more detail in just a bit.

Note: You don't need to provide an ID. The system will generate the ID if your type extension is accepted. For more on entity types, and how they are constructed, check out Module 1, Lesson 6.

Back

Next

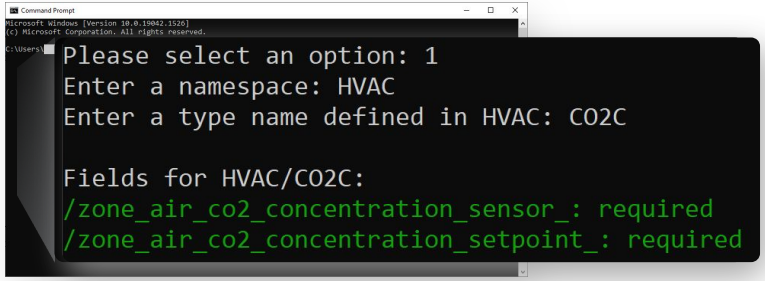
Google

Creating an abstract type

Here are the steps for creating an abstract type outlined in an example.

Example

In the previous example, we explored the ontology and learned that **FAN_SS_CO2C** does essentially what we want, but with CO_2 rather than radon. Therefore, we will want to create a similar abstract type as **CO2C**, using it as a template and modifying its contents to meet our new need.



We can see the definition in the ontology for **CO2C** under HVAC/entity_types/ABSTRACT.yaml.

```
CO2C:
  guid: "91bc028c-acbe-11ed-afa1-0242ac120002"
  description: "Carbon dioxide control."
  is_abstract: true
  implements:
  - OPERATIONAL
  uses:
  - zone_air_co2_concentration_sensor
  - zone_air_co2_concentration_setpoint
```

Back

*Click **Next** to continue the example.*

Next

Creating an abstract type (continued)

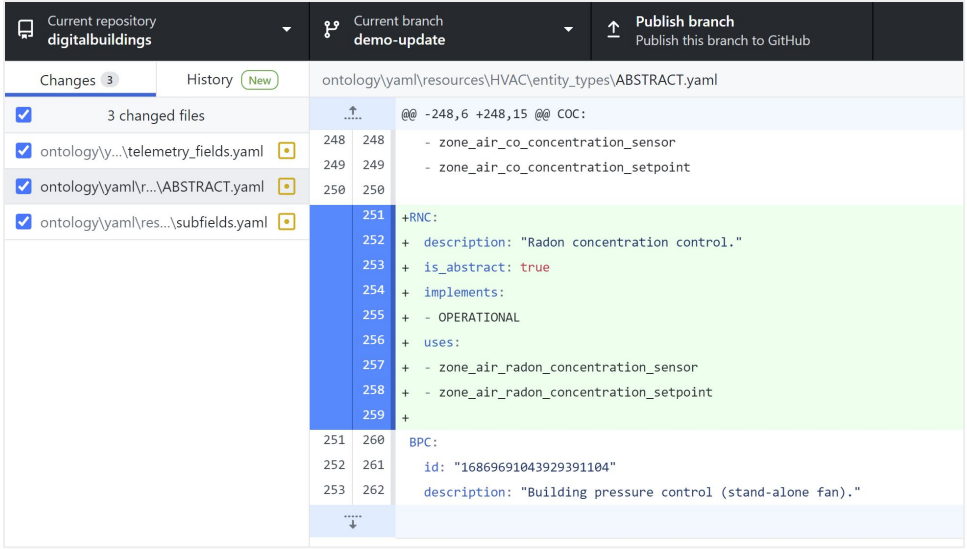
Here are the steps for creating an abstract type outlined in an example.

Example (continued)

Copying the definition for co2c, we simply replace “co2” with “radon” and now have a valid definition. We provide only the description, the fields, and any other necessary information (such as “is_abstract: true”).

Checking our PR in the user interface, we can see it’s now there as well.

Next, we will create the canonical type which uses this new abstract concept and finalizes the field coverage.



Back

Note: You don’t need to provide an ID. The system will generate the ID if your type extension is accepted. For more on entity types, and how they are constructed, check out Module 1, Lesson 6.

Next

Creating a canonical type

Here are the steps for creating a new canonical type.

1. Make sure a comparable type doesn't already exist.
2. Carefully build your type.

   If you'll need a new subfield, field, or abstract type to create your new type, be sure to create them first. Also, avoid applying fields directly to canonical types. Fields should be applied to canonical types through the use of abstract types unless absolutely necessary.

1. Ensure that your type includes the following information:
   - A name that follows conventions.
   - A description.
   - A flag that it is canonical.
   - A list of what the type implements.
1. Create a new Git branch and give the branch a descriptive name.
2. Go to the correct namespace for your new type and update the proper .yaml file.
3. Submit a pull request.

   We'll cover this step in more detail in just a bit.

Note: You don't need to provide an ID. The system will generate the ID if your type extension is accepted. For more on entity types, and how they are constructed, check out Module 1, Lesson 6.

Back

Next

Google

Creating a canonical type

Here are the steps for creating a canonical type outlined in an example.

Example

Now we create the final type. We can copy **FAN_SS_CO2C**, which is close to what we are expecting this type to be, and we replace the **CO2C** abstract type with **RNC**, the new abstract type we just created.

As discussed in previous lessons, the alarm is not necessary. However, if you did want to include it, you could add this as a field directly to the canonical type. We'll examine how in the next slide.

```
FAN_SS_CO2C:
  guid: "91bc028c-acbe-11ed-afa1-0242ac120000"
  description: "CO2 control fan."
  is_canonical: true
  implements:
  - FAN
  - SS
  - CO2C

FAN_SS_RNC:
  description: "Radon control fan."
  is_canonical: true
  implements:
  - FAN
  - SS
  - RNC
```

Back

*Click **Next** to continue the example.*

Next

Google

Creating a canonical type (continued)

Here are the steps for creating a canonical type outlined in an example.

Example (continued)

To add an alarm, we first need to determine the field. Searching telemetry_fields.yaml for the keyword “alarm,” we can find this set of alarms:

```
fabric_protection_alarm
  - ACTIVE
  - INACTIVE
failed_alarm
  - ACTIVE
  - INACTIVE
filter_alarm
  - ACTIVE
  - INACTIVE
master_alarm
  - ACTIVE
  - INACTIVE
```

We see that the **`failed_alarm`** exists, so we could add that to the **`FAN_SS_RNC`** definition in this way:

```
FAN_SS_RNC:
  description: “Radon control fan.”
  is_canonical: true
  implements:
  - FAN
  - SS
  - RNC
  opt_uses:
  - failed_alarm
```

We keep it as optional (**`opt_uses`**) because we don’t require it in order to use this canonical type.

Back

*Click **Next** to continue the example.*

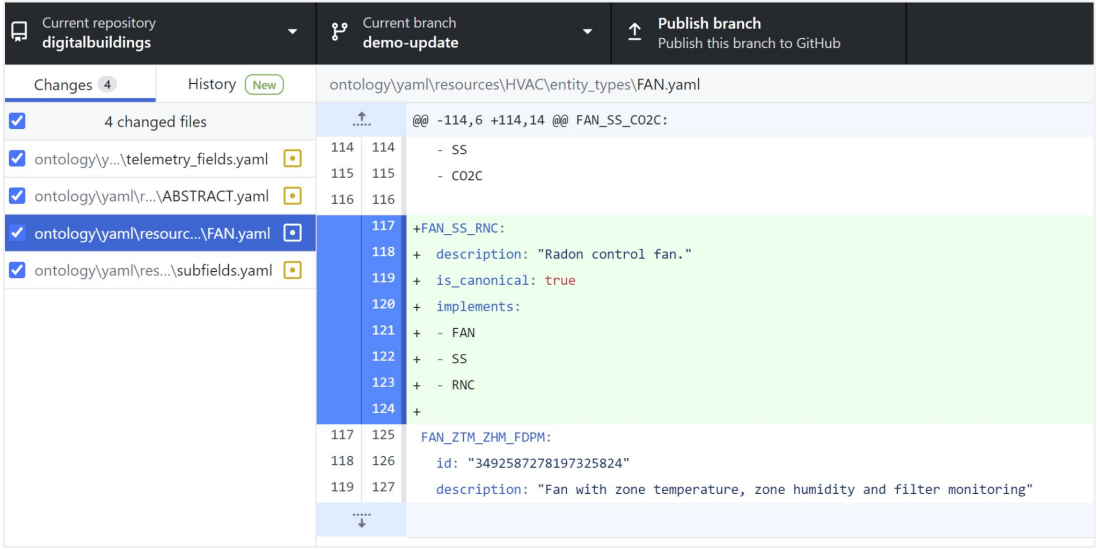
Next

Creating a canonical type (continued)

Here are the steps for creating a canonical type outlined in an example.

Example (continued)

Proceeding forward with the example, we will omit the alarm and add the updated canonical type to the PR.



Back

*Click **Next** to continue the example.*

Next

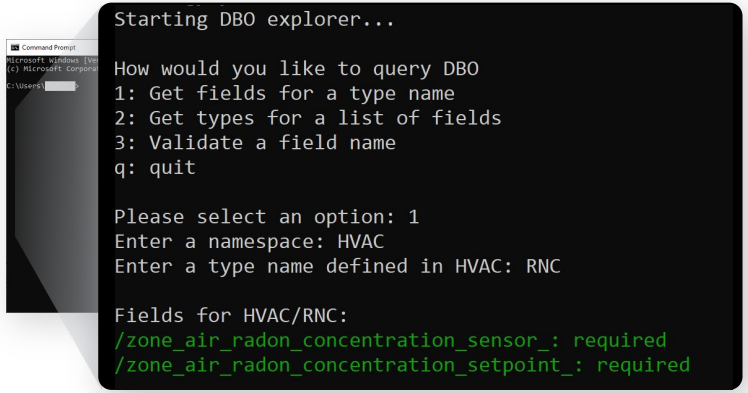
Creating a canonical type (continued)

Let's confirm this update covers the new type. We can do this by exploring the modified local ontology again.

Example (continued)

	A	B	C	D	E	F
1	**Equpiment Name**	**Point Name**	**Units**	**Description**	**Entity Type**	**Field**
2	EF-1	radon_lvl	PPM	Detected radon level.	FAN_SS_RNC	zone_air_radon_concentration_sensor
3	EF-1	radon_lvl_stpt	PPM	Radon level setpoint; threshold where the fan turns on and off.		zone_air_radon_concentration_setpoint
4	EF-1	fan_ss	NO-UNITS	Fan command to run		run_command
5	EF-1	fan_sts	NO-UNITS	Fan feedback, indicating it is running.		run_status
6	~~EF-1~~	~~fan_alarm~~	~~NO-UNITS~~	~~Fan alarm, indicating it has failed.~~		

Running the Ontology Explorer on our local repo (where we've made the modifications) we can explore and see that the new types are defined and support what we want: `RNC` is now defined.



Back

*Click **Next** to continue the example.*

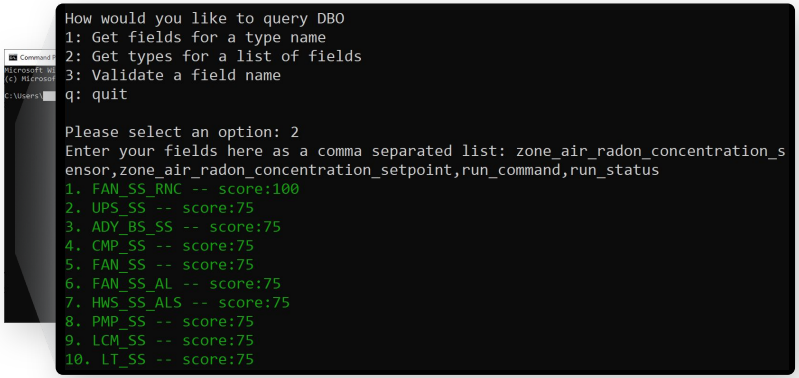
Next

Creating a canonical type (continued)

Let's confirm this update covers the new type. We can do this by exploring the modified local ontology again.

Example (continued)

Let's check that the field set we have for the example exhaust fan now matches the new type **`FAN_SS_RNC`**.



We see in match #1 that **`FAN_SS_RNC`** has a score of 100%. It's a perfect match.

```
MATCH SCORE: 100
MATCHED TYPE: FAN_SS_RNC

ACTUAL FIELDS                          TYPE FIELDS                            OPTIONALITY
=====================================  =====================================  ============
/zone_air_radon_concentration_sensor   /zone_air_radon_concentration_sensor   Required
/run_command                           /run_command                           Required
/zone_air_radon_concentration_setpoint /zone_air_radon_concentration_setpoint Required
/run_status                            /run_status                            Required
                                       /flowrate_capacity                     Optional
                                       /manufacturer_label                    Optional
                                       /current_sensor                        Optional
                                       /powerfactor_sensor                    Optional
                                       /power_capacity                        Optional
                                       /model_label                           Optional
                                       /power_sensor                          Optional
```

Now that we have confirmed the type does what we want, we are ready to push the pull request.

Back

Note: You don't need to provide an ID. The system will generate the ID if your type extension is accepted. For more on entity types, and how they are constructed, check out Module 1, Lesson 6.

Next

Submitting a pull request

Once you've finished adding your proposed ontology extensions to their proper .yaml files, it's time to submit your proposal as a Git pull request.

1. **Commit your work to the branch you created.**

   Include a comment with a brief summary of what you've added.

2. **Push your changes up to GitHub.**

3. **Submit a pull request (PR).**

   If your changes are in a fork, when creating a PR, make sure the "Allow edits by maintainers" checkbox remains checked. This allows the Ontology Validator to automatically write GUIDs to any new entity types.

4. **Fix any issues the validator discovers, then repeat steps 1 and 2.**

   The Ontology Validator will automatically check your proposal for instances where it breaks rules or could cause an incompatibility. It also automatically adds GUIDs to any new entity types. Your proposal will not be reviewed until it can pass validator testing. We'll look more closely at validator testing in a moment.

Back

Note: If you're new to Git, check out their resources to learn how to perform the tasks discussed above. https://git-scm.com/doc

Next

Google

Submitting a pull request

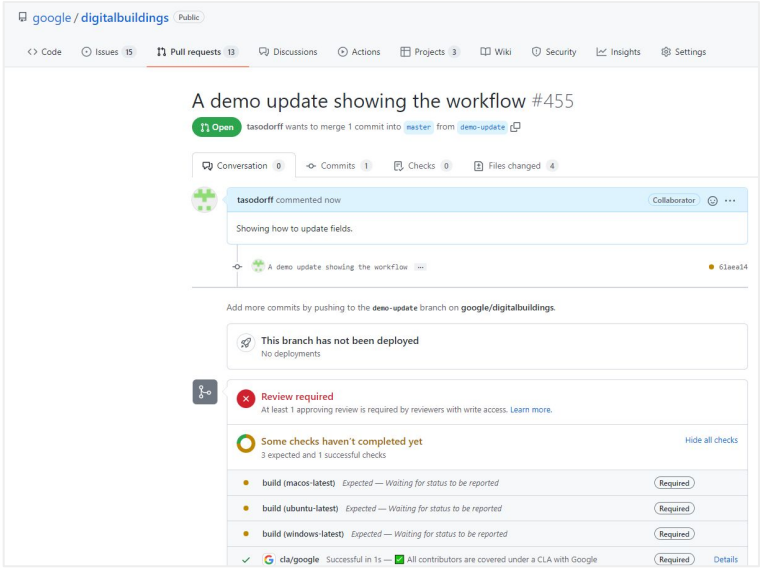
After pushing the PR, we can see it in the web browser.

Example

Here you can see that the validator is running. You must inspect the output once it has completed, looking for errors.

Keep in mind:

1. If this is your first time contributing to the project, the validator will not run automatically. The DBO GitHub admin team will run it for you.
2. If this is your first time contributing to the project, you'll also need to submit a contributor license agreement in order for the PR to be accepted. Follow the instructions on GitHub to sign this.



Back

Next

Validating your extension proposal

Before your extension proposal can be considered, it has to pass the Ontology Validator.

There are two ways to use the Ontology Validator.

1. **Submit a pull request.**

   Whenever you submit a pull request, the Ontology Validator reviews your changes automatically and reports back any issues it detects. You'll need to check the validator outputs when you submit a pull request (if you don't, you will get a message from us that your work didn't pass). If this is your first pull request, the validator will need to be kicked off by an admin on the GitHub repo.

2. **Download the Ontology Validator and run it locally on your machine.**

   For more on how to use the Ontology Validator locally, check out ontology_validator.

Remember, your extension proposal won't be considered until it can pass the validator, so you must correct any issues it finds.

Back

Next

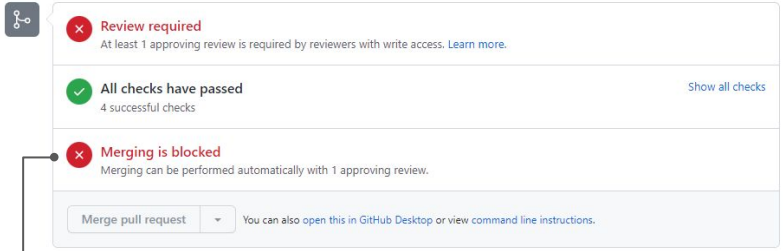
Google

Validating your extension proposal

Here are the steps for validating your extension proposal outlined in an example.

Example

Let's take a look at an example validator output.



Note: Merging is blocked here because the PR has not been approved by someone on the DBO approval team. Once the PR has been reviewed, you'll either get feedback on how to alter your PR to conform more closely to the intent of the ontology or it will get approval for merger. The merger will be handled by the DBO team. You are only responsible for making sure the PR gets approved.

They appear to have passed, but let's inspect to be sure.

Back

*Click **Next** to continue the example.*

Next

Validating your extension proposal (continued)

Here are the steps for validating your extension proposal outlined in an example.

Example (continued)

Click on any of the validator builds and inspect the outputs.

After inspecting the ontology validator outputs, we find that there are no errors, and the warnings are not relevant to our pull request.

The Github admin team can review the PR on its merits and no further MSI work is needed for extending the ontology.

Back

*Click **Next** to continue the example.*

Next

Validating your extension proposal (continued)

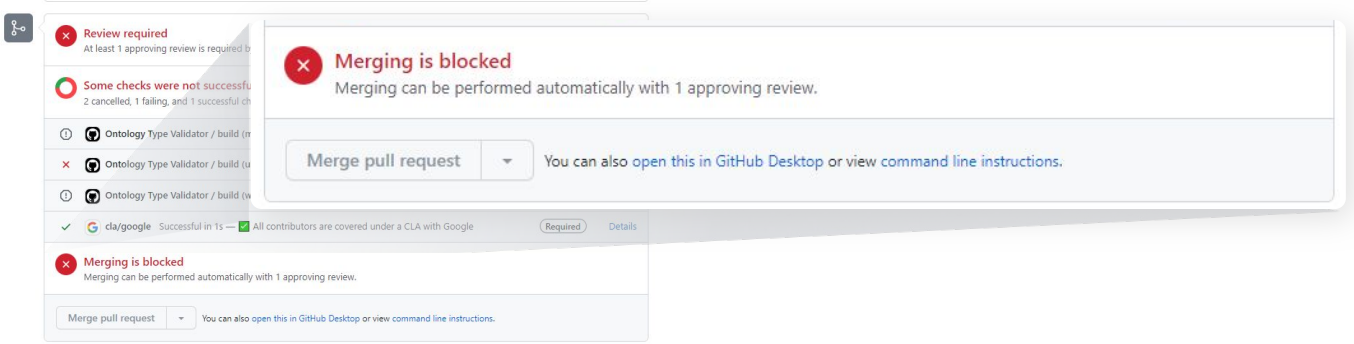
Let's modify the example to show what it looks like to have an error.

Example (continued)

Imagine that we forgot to define radon as a new subfield, but created fields and types that reference it.

```
165     process: "Act of processing."
166     production: "The loop in a system that is responsible for the conditioning of fluid."
167     program: "Embedded, encoded instructions for the automatic performance of a task; typically within a controller."
168     protection: "Act of preventing damage to object."
169     # radon: "The chemical element number 86, a rare radioactive gas belonging to the noble gas series."
170     rain: "Liquid water in the form of droplets that have condensed from atmospheric water vapor."
171     recovery: "Component or process used for the reclamation of heat."
172     red: "Red light fracture of ambient light"
173     release: "To free from constraints (e.g. to release the door from its magnetic lock)."
```

Let's resubmit the PR and see what errors we get.



We can see that the merge is now blocked because of the validator failure.

Back

*Click **Next** to continue the example.*

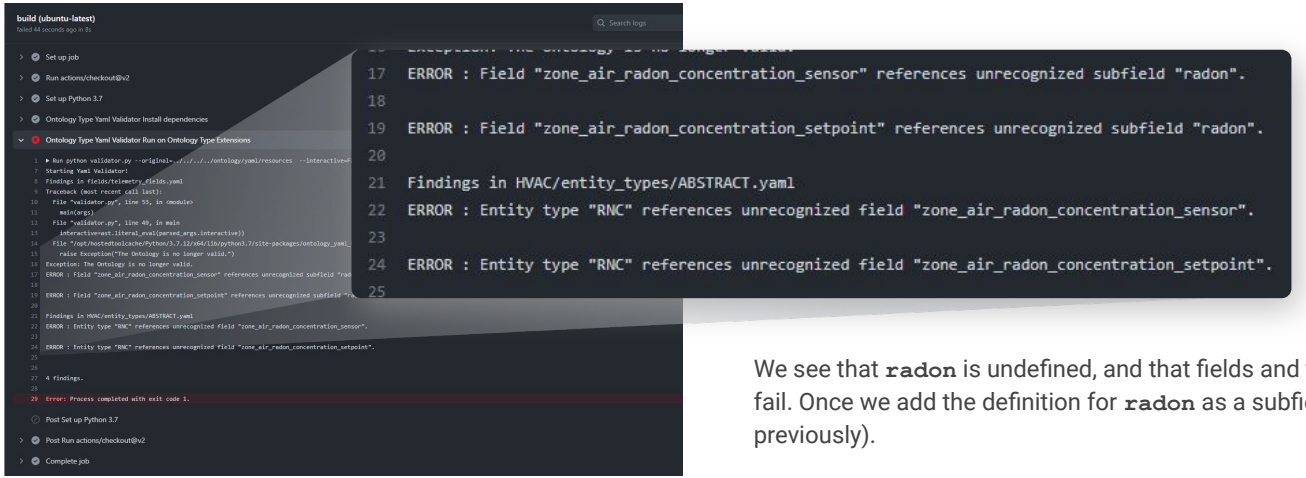
Next

Validating your extension proposal (continued)

Let's modify the example to show what it looks like to have an error.

Example (continued)

Let's explore the error messages.



We see that **`radon`** is undefined, and that fields and types that use that field subsequently fail. Once we add the definition for **`radon`** as a subfield the errors will go away (as shown previously).

Back

Next

Accepting ontology extensions

Once your extension proposal passes validator testing, the Digital Buildings team will examine your proposal.

The Digital Buildings team will examine your proposal with both an eye for how it applies to the existing ontology and what impact it could have on the ontology long-term.

The team might send you questions or proposed changes.

If so, addressing these issues as quickly as possible is the best way to ensure that your proposal will be accepted in a timely fashion.

Once all questions and issues have been addressed, the team will make a final decision on whether your proposed extensions should be included in the ontology.

Back

Next

Google

Lesson 5

Knowledge check

Let’s take a moment to reflect on what you’ve learned so far.

- The next slides will have questions about the concepts that were introduced in this lesson.
- Review each question and select the correct response.
- After this knowledge check, you’ll wrap up Lesson 5.

You won’t be able to move forward until the correct answer is selected.

Back

*Click **Next** when you’re ready to begin.*

Next

Google

Knowledge check 1

You're working on an ontology extension proposal that introduces two new non-abstract types. Both types use four new fields, along with several new subfields.

How should you go about submitting your extension proposal?

Select the best answer from the options listed below.

- Start by submitting the new subfields. Once they are approved, submit the new fields. After the fields are approved, submit the types.
- Submit all the new material at once, so reviewers can see the new items with full context.
- Submit the fields and subfields together, then the types, so that all non-canonical information stays together.
- It depends on the context.

Back

Next

Google

Knowledge check 1

You're working on an ontology extension proposal that introduces two new non-abstract types. Both types use four new fields, along with several new subfields.

How should you go about submitting your extension proposal?

Select the best answer from the options listed below.

- Start by submitting the new subfields. Once they are approved, submit the new fields. After the fields are approved, submit the types.
- Submit all the new material at once, so reviewers can see the new items with full context.
- Submit the fields and subfields together, then the types, so that all non-canonical information stays together.
- It depends on the context.

Close... but not quite right! 🤔

Think back to the examples seen earlier in this lesson. Was it done in one pull request or multiple pull request? What was the reasoning for the approach that was taken?

Try again

Back

Next

Knowledge check 1

You're working on an ontology extension proposal that introduces two new non-abstract types. Both types use four new fields, along with several new subfields.

How should you go about submitting your extension proposal?

Select the best answer from the options listed below.

- Start by submitting the new subfields. Once they are approved, submit the new fields. After the fields are approved, submit the types.
- Submit all the new material at once, so reviewers can see the new items with full context.
- Submit the fields and subfields together, then the types, so that all non-canonical information stays together.
- It depends on the context.

Close... but not quite right! 🤔

Think back to the examples seen earlier in this lesson. Was it done in one pull request or multiple pull request? What was the reasoning for the approach that was taken?

Try again

Back

Next

Knowledge check 1

You're working on an ontology extension proposal that introduces two new non-abstract types. Both types use four new fields, along with several new subfields.

How should you go about submitting your extension proposal?

Select the best answer from the options listed below.

- Start by submitting the new subfields. Once they are approved, submit the new fields. After the fields are approved, submit the types.
- Submit all the new material at once, so reviewers can see the new items with full context.
- Submit the fields and subfields together, then the types, so that all non-canonical information stays together.
- It depends on the context.

Close... but not quite right! 🤔

Think back to the examples seen earlier in this lesson. Was it done in one pull request or multiple pull request? What was the reasoning for the approach that was taken?

Try again

Back

Next

Knowledge check 1

You're working on an ontology extension proposal that introduces two new non-abstract types. Both types use four new fields, along with several new subfields.

How should you go about submitting your extension proposal?

Select the best answer from the options listed below.

Start by submitting the new subfields. Once they are approved, submit the new fields. After the fields are approved, submit the types.

Submit all the new material at once, so reviewers can see the new items with full context.

Submit the fields and subfields together, then the types, so that all non-canonical information stays together.

It depends on the context.

That's right! 🎉

It depends on how many extensions you want to make. Sometimes it will be perfectly fine to submit one PR containing all changes, such as when you are proposing very small additions to a particular type. Sometimes it may be more effective to do proposals in several batches to prevent tedious rework. You will need to use your judgment to determine what is most appropriate for your project.

Back

Next

Knowledge check 2

When should you check your extension proposal with the validator tool?

Select the best answer from the options listed below.

- Locally, before you submit your pull request
- Automatically, after you submit your pull request
- Either locally or automatically

Back

Next

Google

Knowledge check 2

When should you check your extension proposal with the validator tool?

Select the best answer from the options listed below.

- Locally, before you submit your pull request
- Automatically, after you submit your pull request
- Either locally or automatically

Close... but not quite right! 🤔

Don't forget that your pull request will be validated once you push to GitHub.

Try again

Back

Next

Google

Knowledge check 2

When should you check your extension proposal with the validator tool?

Select the best answer from the options listed below.

Locally, before you submit your pull request

Automatically, after you submit your pull request

Either locally or automatically

Close... but not quite right! 🤔

Don't forget that you can validate locally using the validator, too.

Try again

Back

Next

Google

Knowledge check 2

When should you check your extension proposal with the validator tool?

Select the best answer from the options listed below.

Locally, before you submit your pull request

Automatically, after you submit your pull request

Either locally or automatically

That's right! 🎉

The key thing to remember is that your extension proposal has to pass the validator before it will be considered by the Digital Buildings team for inclusion. Whether you test your work locally as you go using the ontology validation tool, or wait until you submit your proposal for the automatic validation test is up to you.

Back

Next

Google

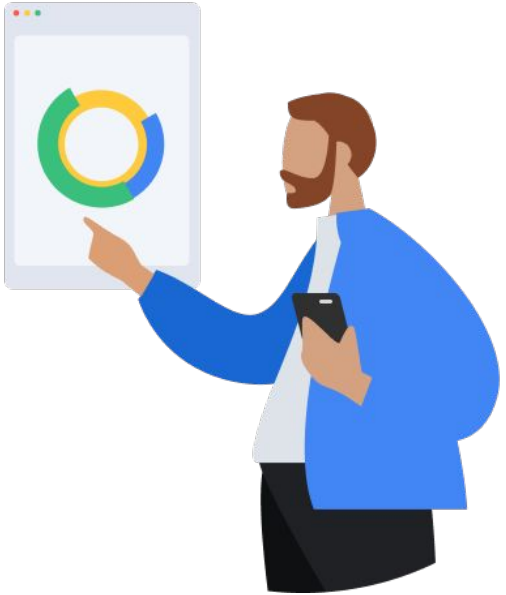
Lesson 5 summary

Let's review what you learned about:

- Ontology extensions

Now you should be able to:

- Submit a pull request to propose an ontology extension for:
  - Subfields
  - Fields
  - Abstract types
  - Non-abstract types



Back

Next

Google

You completed Lesson 5!

Now's a great time to take a quick break before starting Lesson 6.

Ready for Lesson 6?

Let's go!

Back

*Press the **Esc** key on your keyboard to exit presentation mode.*

Google

Have questions?

For future reference, keep these contacts and resources easily accessible for technical and procedural questions.

Key contacts

- For project-related questions: Your project's TPM or DBC
- For DBO questions: Trevor (tsodorff@) or Digital Buildings Discussions

Helpful resources

Bookmark these resources for future reference.

- Digital Buildings Project GitHub
  Contains source code, tooling, and documentation for the DBO.
- Git Documentation
  Provides tips and guidance on how to use Git.
- BOS for everyone site
  Provides additional information about the BOS program.