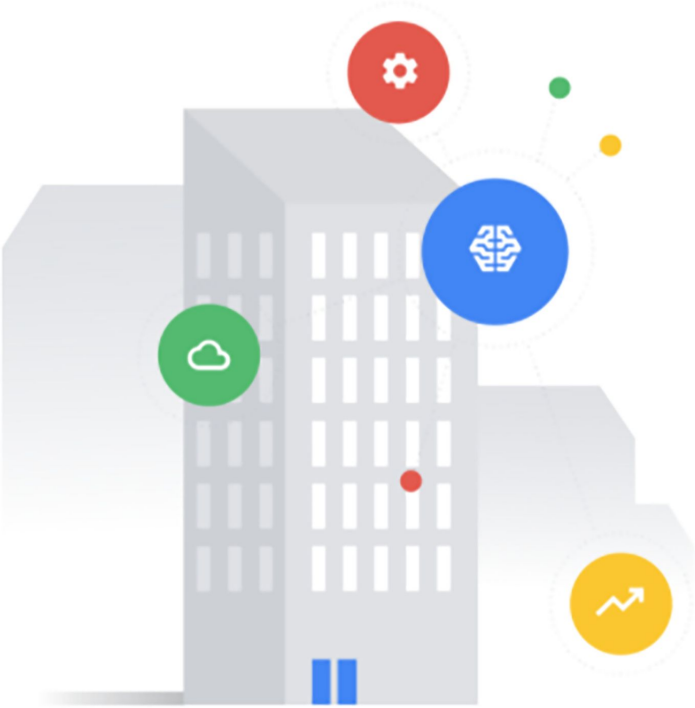
Google

Module 2 | **Lesson 7**

Data modeling with the DBO



Before you get started

This learning module 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 content.

1 View this content outside of GitHub.

- For the best learning experience, you're encouraged to download a copy so links and other interactive features will be enabled.
- To download a copy of this lesson, click **Download** in the top-right corner of this content block.
- After downloading, open the file in your preferred PDF reader application.

2 Navigate by clicking the buttons and links.

- For the best learning experience, using your keyboard or mouse wheel to navigate is discouraged. However, this is your only option if you're viewing from GitHub.
- If you're viewing this content outside of GitHub:
  - 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.

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 sixth step of data modeling with the DBO.

1. Determine which devices need to be modeled
2. Determine which data points are required
3. Name each data point using the DBO
4. Propose an ontology extension
5. Construct and finalize the building configuration file
6.
7. Complete!

Validate the instance and telemetry

As you construct the building config, it's critical to ensure it fully conforms to the DBO. You'll need to validate the instance and the mapped telemetry frequently before completing it for your project.

Back

Next

Lesson 7

Validate the instance and telemetry

What you'll learn about:

- The Instance Validator tool
- Instance validation
- Telemetry validation
- Validation feedback

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

- Run the Instance Validator with basic commands and additional parameters.
- Generate a report file to share validation results with others.
- Identify the root cause of validation errors and correct them.
- Enable telemetry validation mode in the Instance Validator.

Back

Next

Google

Instance Validator

The **Instance Validator** is a tool used to validate building configuration files for conformance with the DBO and to validate end-to-end connectivity.

Command Prompt

```
C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python
instance_validator.py --input C:\[redacted]\Desktop\OnboardDemo\demo.yaml

Starting validator...


Starting universe generation...


Starting config validation...


Loading config files...

Validating syntax please wait ...
Opening file: C:\[redacted]\Desktop\OnboardDemo\demo.yaml, please wait ...

Starting config validation...

Validating entities ...
All entities validated

C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>
```

Back

Next

Google

Instance Validator

What's the Instance Validator?

The **Instance Validator** is used to check if your building config is formatted properly and the DBO was applied accurately. It ensures the building telemetry sent to the cloud aligns with what you've represented in the concrete model. In other words, it makes sure your building config actually works.

You'll use the Instance Validator to validate the instance and telemetry of a building config.

The Instance Validator is part of the Digital Buildings Project. Installation and usage information can be found in instance_validator in the GitHub repo.

Why validate a building config file?

Validation identifies problems in a building config.

We encourage incorporating iterative validation into your data modeling workflow to identify and fix errors while you're constructing the building config. This approach allows you to address errors in smaller batches.

Waiting to validate until you've completed the entire building config file could result in a large batch of errors (and a lot of tedious rework).

Instance Validator (continued)

What's needed to run the Instance Validator?

Before attempting to validate the instance of a building config, you'll need:

- [x] A building config file (complete or partially complete)
- [x] A version of Python installed on your machine (see python.org)
- [x] The Instance Validator tool installed on your machine (see instructions)
- [x] The Ontology Validator installed on your machine (see instructions)

The Instance Validator is also used for telemetry validation, which has a few additional requirements. These will be covered later in this lesson. For now, let's focus on instance validation with the Instance Validator.

Back

Next

Instance validation with the Instance Validator

The **Instance Validator** can be run at any time while you're translating the building config.

Command Prompt

```
python instance_validator.py input --input
path/to/YOUR_BUILDING_CONFIG.yaml
```

Note: The Instance Validator is written in Python and takes as an argument the path pointing to the instance files.

Back

Next

Google

Instance validation with the Instance Validator

Basic command

You'll run the Instance Validator from your machine's terminal or command prompt using the basic command:

```
python instance_validator.py input --input
path/to/YOUR_BUILDING_CONFIG.yaml
```

By default, this basic command will:

- Validate a single building config against the current version of the DBO.
- Write validation results in standard out.

It's important that the `--input` parameter points to the correct path to the building config file to be validated.

Additional parameters

The following parameters can also be used:

Multiple `-- input path/to/YOUR_BUILDING_CONFIG.yaml`
Validates multiple building configs at the same time. You can use as many `--input` parameters for as many building configs you'd like to validate at one time.

`-- report-filename path/to/REPORT-NAME.txt`

Provides the validation results in a separate report file. You'll need to specify your desired location and name for the report file. Report files are needed if you want to share results with another person.

`--modified-ontology-types path/to/modified/ontology/types/folder`

Validates the building config against a local version of a modified DBO. This is handy if you've extended the ontology by adding new modeling concepts to your local ontology but haven't yet submitted them to the DBO or your PR is not yet merged.

Back

Next

Lesson 7

Practice 1

Let's take a moment to apply what you've learned so far.

- The next slide will give you an opportunity to use the Instance Validator to validate a sample building configuration file.
- You'll use the sample building config file named "demo.yaml" from the Lesson 7 practice folder.
- If you haven't done so already, install the Instance Validator on your machine.
- After this practice activity, you'll move on to instance validation feedback.

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

Back

Next

Google

Practice 1

Let's say you have a building configuration file that needs to be validated.

Use the Instance Validator to validate the instance of the building config "demo.yaml".

Go to the Lesson 7 practice folder to download the file "demo.yaml" to your machine. Then, follow the steps displayed on the right.

Note: In order to validate for this practice activity, you must have the Instance Validator already installed on your machine.

Steps

From your terminal or command prompt, run Instance Validator using the following command:

```
python instance_validator.py --input
path/to/YOUR_BUILDING_CONFIG.yaml
```

Make sure the `--input` parameter points to the correct path to the sample building config file.

Back

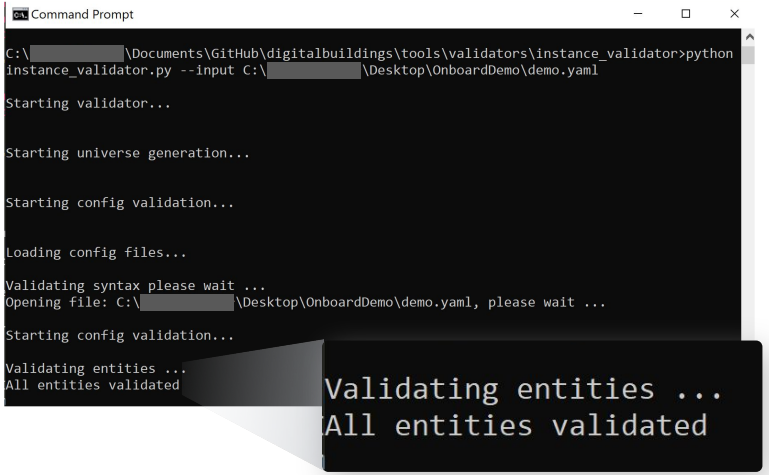
*When you're ready, click **Next** to check your work.*

Next

Practice 1

Check your work! 👨‍🏫

The validation results should have been successful.
Did you end up with this success message?



Back

Keep this file easily accessible for the next activity.
*Click **Next** to move on to the next practice activity.*

Next

Lesson 7

Practice 2

Let's continue to apply what you've learned so far.

- The next slide will give you an opportunity to use the Instance Validator to generate a report file of validation results.
- You'll continue to use the sample building config file named "demo.yaml" from the Lesson 7 practice folder.
- If you haven't done so already, install the Instance Validator on your machine.
- After this practice activity, you'll move on to instance validation feedback.

Back

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

Next

Google

Practice 2

Let's say you need to share the validation results with someone else on your team.

Use the Instance Validator to generate a report file for the building config "demo.yaml".

Go to the Lesson 7 practice folder to download the file "demo.yaml" to your machine. Then, follow the steps displayed on the right.

Note: In order to validate for this practice activity, you must have the Instance Validator already installed on your machine.

Steps

From your terminal or command prompt, run Instance Validator using the following command:

```
python instance_validator.py --input
path/to/YOUR_BUILDING_CONFIG.yaml --report-filename
path/to/REPORT-NAME.txt
```

Make sure the `--input` parameter points to the correct path to the sample building config file.

Also make sure the `--report-filename` parameter specifies your desired location and name for the report file.

Back

*When you're ready, click **Next** to check your work.*

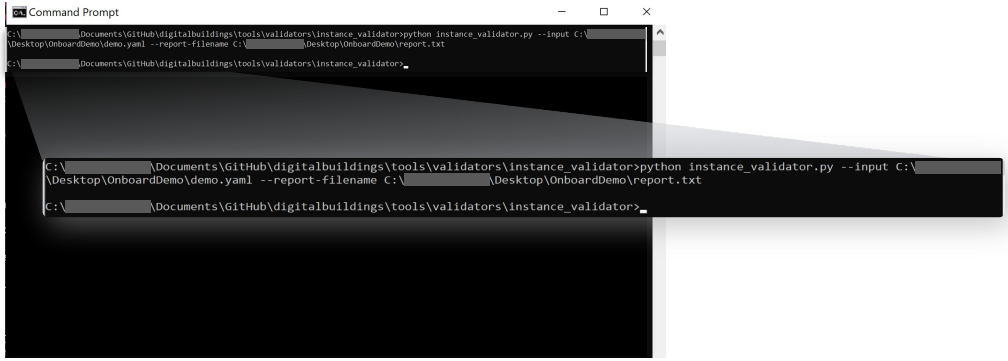
Next

Practice 2

Check your work! 👩‍🏫

A new report file should have been generated and can be inspected for results.
Is it in your specified location? Does it have the file name you specified?

If you check your terminal or command prompt, you'll see that the validation results didn't print to standard out. You'll need to open the report file to see if the validation was successful or if it failed.



report.txt

Here's a look of the opened the report file showing the validation results..

```
Starting validator...


Starting universe generation...


Starting config validation...


Loading config files...

Validating syntax please wait ...
Opening file: C:\Users\[redacted]\OnboardDemo\demo.yaml, please wait ...

Starting config validation...

Validating entities ...
All entities validated
```

Back

*Click **Next** to move on to instance validation feedback.*

Next

Instance validation feedback

The Instance Validator will show whether the validation is successful or resulted in an error.

Success message

Here's an example of an instance validation that resulted in a success. This is what you're striving for after each validation iteration and upon completion of a building config file.

```
Command Prompt

C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python
instance_validator.py --input C:\[redacted]\Desktop\OnboardDemo\demo.yaml

Starting validator...

Starting universe generation...

Starting config validation...

Loading config files...

Validating syntax please wait ...
Opening file: C:\[redacted]\Desktop\OnboardDemo\demo.yaml, please wait ...

Starting config validation...

Validating entities ...
All entities validated
```

```
Validating entities ...
All entities validated
```

Error message

Here's an example of an instance validation that resulted in an error. If this occurs, you'll need to identify the root cause of the error, fix it, and re-run validation to confirm the error is resolved.

```
Command Prompt

C:\[redacted]\Documents\GitHub\
instance_validator.py --input C:\

Starting validator...

Starting universe generation...

Starting config validation...

Loading config files...

Validating syntax please wait ...
Opening file: C:\[redacted]\Des

Starting config validation...

Validating entities ...
```

```
Validating entities ...
Config must contain a non-deleted entity with a building type
Traceback (most recent call last):
  File "C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_valid
>
    handler.RunValidation(
  File "C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_valid
tion
    entities = _ValidateConfig(filenames, universe)
  File "C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_valid
onfig
    return helper.Validate(entities, config_mode)
  File "C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_valid
    raise SyntaxError('Building Config must contain an '
SyntaxError: Building Config must contain an entity with a building type
```

Back

Note: In the next set of practice activities, we'll look at common validation errors and tips for fixing them.

Next

Lesson 7

Practice 3

Let's take a moment to apply what you've learned so far.

- The next slides will give you a few opportunities to use the Instance Validator to validate the sample building configuration files from the Lesson 7 practice folder.
- If you haven't done so already, install the Instance Validator on your machine.
- After this practice activity, you'll move on to complete another activity to continue practicing with the Instance Validator.

Back

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

Next

Google

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the Lesson 7 practice folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Steps

From your terminal or command prompt, run Instance Validator using the following command:

```
python instance_validator.py --input
path/to/YOUR_BUILDING_CONFIG.yaml
```

Make sure the `--input` parameter points to the correct path to the sample building config file.

Back

Note: In order to validate for this practice activity, you must have the Instance Validator already installed on your machine.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

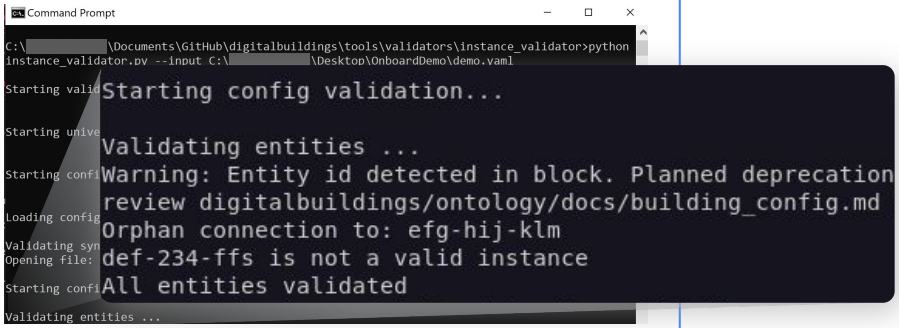
When you're ready, select a building config file you validated to check your work.

- demo_bad_connection.yaml
- demo_missing_building.yaml
- demo_bad_field.yaml
- demo_missing_keys.yaml
- demo_bad_indent.yaml
- demo_missing_type.yaml
- demo_bad_state.yaml
- demo_missing_value.yaml

Check your work! 👨‍🏫

The validation of the building config "demo_bad_connection.yaml" should have resulted in an error. **Did you end up with this error message?** Click **Next** for info about the error, root cause, and solution.

Terminal



Back

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

- demo_bad_connection.yaml
- demo_missing_building.yaml
- demo_bad_field.yaml
- demo_missing_keys.yaml
- demo_bad_indent.yaml
- demo_missing_type.yaml
- demo_bad_state.yaml
- demo_missing_value.yaml

Back

Check your work! 👨‍🏫 (continued)

Here's the mistake and how to fix it.

Error

```
Orphan connection to: efg-hij-klm
```

Root cause

The entity `efg-hij-klm` isn't defined in the building config.

Solution

Fix the affected connection. Either replace the zone with another entity defined in the building config or define a new entity for the zone.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

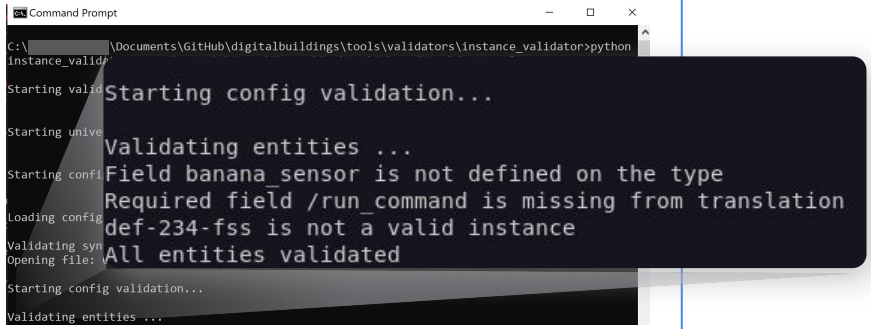
When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Check your work! 👩‍🏫

The validation of the building config "demo_bad_field.yaml" should have resulted in an error. **Did you end up with this error message?**
Click **Next** for info about the error, root cause, and solution.

Terminal



Back

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👩‍🏫 (continued)

Here's the mistake and how to fix it.

Error

1. `Field banana_sensor is not defined on the type`
2. `Required field /run_command is missing from translation`

Root cause

1. `banana_sensor` isn't a valid field in the Digital Buildings Ontology.
2. A translation needs a required field added to it.

Solution

1. Remove the field `banana_sensor` or replace it with a valid field.
2. Add the specified required field `run_command` to the affected translation.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

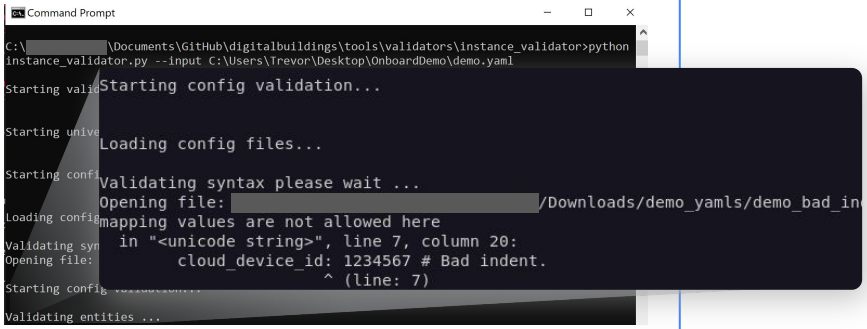
When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Check your work! 👨‍🏫

The validation of the building config "demo_bad_indent.yaml" should have resulted in an error. **Did you end up with this error message?** Click **Next** for info about the error, root cause, and solution.

Terminal



Back

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👩‍🏫 (continued)

Here's the mistake and how to fix it.

Error

The validation message doesn't specify the error, but it does tell you exactly where the error is located: `in "<unicode string>", line 7 column 7:`.

Root cause

Check the building config file to find the root cause. You should see the indent is incorrect due to extra spaces next to `cloud_device_id: 1234567`.

Solution

Delete the extra spaces before `cloud_device_id: 1234567`.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

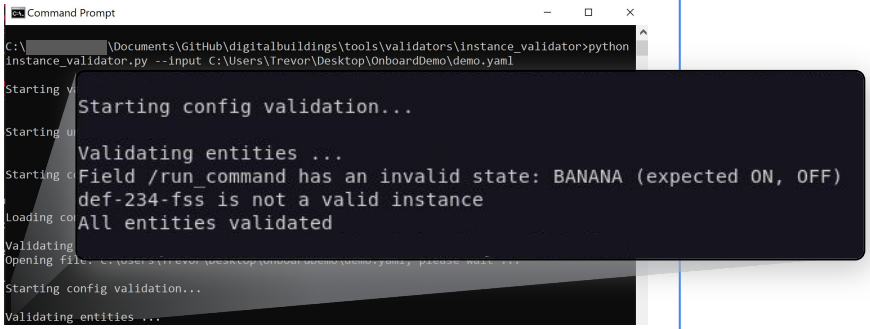
When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Check your work! 👩‍🏫

The validation of the building config "demo_bad_state.yaml" should have resulted in an error. **Did you end up with this error message?**
Click **Next** for info about the error, root cause, and solution.

Terminal



Back

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👩‍🏫 (continued)

Here's the mistake and how to fix it.

Error

`Field /run_command has an invalid state: BANANA (expected ON, OFF)`

Root cause

`BANANA` isn't a valid state in the Digital Buildings Ontology.

Solution

Correct the states defined for the field `run_command`. `BANANA` should be replaced with the states `ON` and `OFF`.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Check your work! 👨‍🏫

The validation of the building config "demo_missing_building.yaml" should have resulted in an error. **Did you end up with this error message?** Click **Next** for info about the error, root cause, and solution.

Terminal

Command Prompt

```
C:\[redacted]\De
instance_validator

Starting validator

Starting universe

Starting config va

Loading config fil

Validating syntax
Opening file: C:\U

Starting config va

Validating entitie
```

```
Starting config validation...

Validating entities ...
Config must contain a non-deleted entity with a building type
Traceback (most recent call last):
  File "[redacted]/Downloads/digitalbuildings/tools/
py", line 112, in <module>
    handler.RunValidation(
  File "[redacted]/Downloads/digitalbuildings/tools/
", line 125, in RunValidation
    entities = _ValidateConfig(filenames, universe, is_udmi)
  File "[redacted]/Downloads/digitalbuildings/tools/
", line 89, in _ValidateConfig
    return helper.Validate(entities, config_mode, is_udmi)
  File "[redacted]/Downloads/digitalbuildings/tools/
", line 260, in Validate
    raise SyntaxError('Building Config must contain an '
SyntaxError: Building Config must contain an entity with a building type
```

Back

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👨‍🏫 (continued)

Here's the mistake and how to fix it.

Error

```
SyntaxError: Building Config must contain an entity with
a building type.
```

Root cause

A building isn't defined in the building config file.

Solution

Define an entity for the building.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

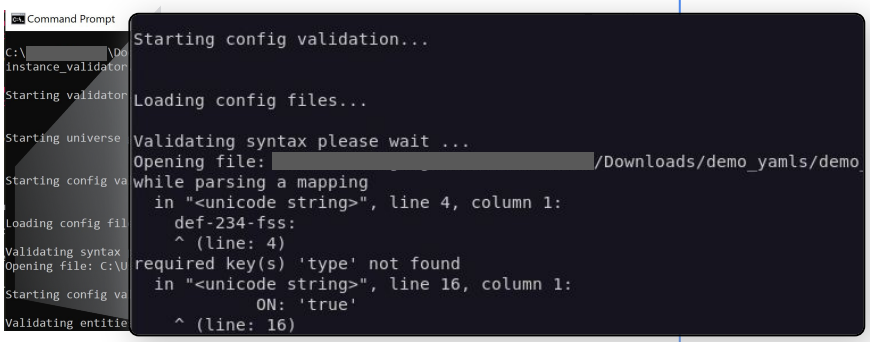
When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Check your work! 👩‍🏫

The validation of the building config "demo_missing_keys.yaml" should have resulted in an error. **Did you end up with this error message?** Click **Next** for info about the error, root cause, and solution.

Terminal



Back

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👩‍🏫 (continued)

Here's the mistake and how to fix it.

Error

`required key(s) 'type' not found`

Root cause

The entity `def-234-fss` doesn't have an entity type.

Solution

Add an entity type to the entity `def-234-fss`.

Wait!

There's actually another error present in this building config. Apply the above solution, and then try validating the file again. Click **Next** for info about the second error.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👩‍🏫
(continued)

After fixing the first error and validating the building config again, you'll see another error message indicating `cloud_device_id` is missing.

Why? Because the Instance Validator will fail on the first error (or common group of errors) it spots. This is why it's important to troubleshoot dynamically and validate in small batches to avoid a lot of hidden errors in subsequent validation runs.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Check your work! 👨‍🏫

The validation of the building config "demo_missing_type.yaml" should have resulted in an error. **Did you end up with this error message?** Click **Next** for info about the error, root cause, and solution.

Terminal

```
Starting config validation...

Loading config files...

Validating syntax please wait ...
Opening file: [redacted]/Downloads/demo_yamls/demo_
while parsing a mapping
  in "<unicode string>", line 4, column 1:
    def-234-fss:
    ^ (line: 4)
required key(s) 'type' not found
  in "<unicode string>", line 19, column 1:
            ON: 'true'
    ^ (line: 19)
```

Back

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👨‍🏫 (continued)

Here's the mistake and how to fix it.

Error

`required key(s) 'type' not found`

Root cause

The entity `def-234-fss` doesn't have an entity type.

Solution

Add an entity type to the entity `def-234-fss`.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👩‍🏫

The validation of the building config "demo_missing_value.yaml" should have resulted in an error. **Did you end up with this error message?** Click **Next** for info about the error, root cause, and solution.

Terminal

Command Prompt

```
C:\[redacted]\Documents\GitH
instance_validator.py --input

Starting validator...

Starting universe generation...

Starting config validation...

Loading config files...

Validating syntax please wait
Opening file: C:\[redacted]

Starting config validation...

Validating entities ...
```

```
Validating syntax please wait ...
Opening file: [redacted]/Downloads/demo_yamls/demo_missi
Invalid Entity def-234-fss: raw_field_name cannot be empty
Traceback (most recent call last):
  File "[redacted]/Downloads/digitalbuildings/tools/vali
py", line 112, in <module>
    handler.RunValidation(
  File "[redacted]/Downloads/digitalbuildings/tools/vali
", line 125, in RunValidation
    entities = _ValidateConfig(filenames, universe, is_udmi)
  File "[redacted]/Downloads/digitalbuildings/tools/vali
", line 86, in _ValidateConfig
    entities, config_mode = Deserialize(filenames)
  File "[redacted]/Downloads/digitalbuildings/tools/vali
", line 71, in Deserialize
    entity = entity_instance.EntityInstance.FromYaml(
  File "[redacted]/Downloads/digitalbuildings/tools/vali
tance.py", line 864, in FromYaml
    translation = _ParseTranslation(entity_yaml[parse.TRANSLATION_KEY])
  File "[redacted]/Downloads/digitalbuildings/tools/vali
tance.py", line 669, in _ParseTranslation
    ft_object = ft_lib.MultiStateValue(std_field_name, raw_field_name,
  File "[redacted]/Downloads/digitalbuildings/tools/vali
slation.py", line 103, in __init__
    super().__init__(std_field_name, raw_field_name)
  File "[redacted]/Downloads/digitalbuildings/tools/vali
slation.py", line 81, in __init__
    raise ValueError('raw_field_name cannot be empty')
ValueError: raw_field_name cannot be empty
```

Next

Practice 3

Let's say you have a building config that needs to be validated, but it has a mistake. Don't worry, it's inevitable!

Use the Instance Validator to validate at least two building config files with errors.

Go to the [Lesson 7 practice]() folder to download two or more files labeled "bad" or "missing". Then, follow the steps displayed on the right.

When you're ready, select a building config file you validated to check your work.

demo_bad_connection.yaml	demo_missing_building.yaml
demo_bad_field.yaml	demo_missing_keys.yaml
demo_bad_indent.yaml	demo_missing_type.yaml
demo_bad_state.yaml	demo_missing_value.yaml

Back

Check your work! 👩‍🏫 (continued)

Here's the mistake and how to fix it.

Error

Sometimes, it isn't entirely clear from the validation message what or where exactly the error is. In these cases, it's up to you and your knowledge of the building configuration format to identify and correct the error.

Root cause

Check the building config file to find the root cause. You should see there's a missing value for the key `present_value`.

Solution

Qualify the corresponding point from the JSON payload.

Take your time and try validating another sample building config.
When you're ready, click **Next** to move on to Practice 4.

Next

Lesson 7

Practice 4

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

- The next slide will give you an opportunity to use the Instance Validator to validate a sample building configuration file and troubleshoot its error.
- You'll use the sample building config file named "demo_test.yaml" from the Lesson 7 practice folder.
- If you haven't done so already, install the Instance Validator on your machine.
- After this practice activity, you'll move on to telemetry validation.

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

Back

Next

Google

Practice 4

Let's say you have a building configuration file that needs to be validated.

Use the Instance Validator to validate the instance of the building config "demo_test.yaml".

Go to the Lesson 7 practice folder to download the file "demo_test.yaml" to your machine. Then, follow the steps displayed on the right.

Note: In order to validate for this practice activity, you must have the Instance Validator already installed on your machine.

Steps

From your terminal or command prompt, run Instance Validator using the following command:

```
python instance_validator.py --input
path/to/YOUR_BUILDING_CONFIG.yaml
```

Make sure the `--input` parameter points to the correct path to the sample building config file.

Back

*When you're ready, click **Next** to review results and identify errors.*

Next

Practice 4

Uh oh! There's an error. 🤔

Here are the validation results. **Do you see the error message?**

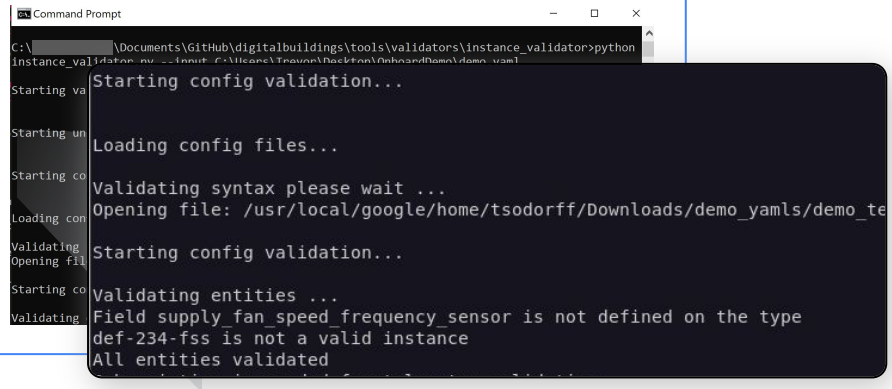
This building config needs to be corrected. You'll need to identify the root cause of the error and fix it in the building config file.

Find and fix the error in the building config "demo_test.yaml". Then, validate it again to confirm the error is fixed.

Use your text editor to modify the file "demo_test.yaml" and re-validate it using the Instance Validator.

Hint: The error message is: `Field supply_fan_speed_frequency_sensor is not defined on the type`

Terminal



Back

*When you're ready, click **Next** to review results and check your work.*

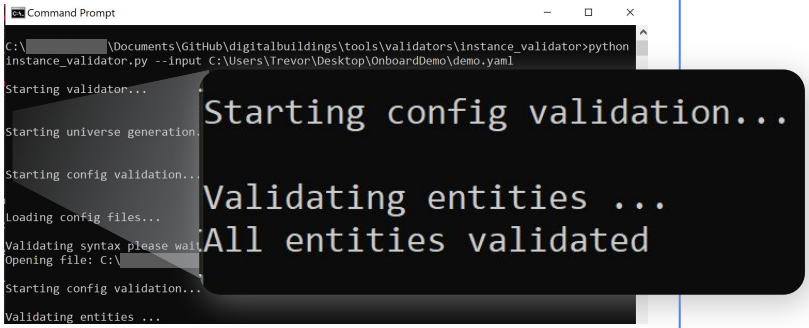
Next

Practice 4

Check your work! 👩‍🏫

After fixing the error, the building config “demo_test.yaml” should have been successfully validated. **Did you get a success message?**

Terminal



What was the error?

The original building config file had a translated field in the entity `def-234-fss` that isn’t associated with the entity type `HVAC/FAN_SS`. The error message specified the problematic field is `supply_fan_frequency_sensor`.

How should it have been fixed?

You would’ve needed to remove the problematic field from the affected translation. If this wasn’t immediately apparent to you, a quick run of the entity type `HVAC/FAN_SS` through the Ontology Explorer would help you confirm that `supply_fan_frequency_sensor` is absent from its associated fields.

Back

*Click **Next** to move on to telemetry validation.*

Next

Telemetry validation with the Instance Validator

The **Instance Validator** is also used for telemetry validation of building configs by enabling “telemetry validation mode”.

Command Prompt

```
python instance_validator.py --input
path/to/YOUR_BUILDING_CONFIG.yaml --
subscription
projects/google.com:your-project/subscriptions/y
our-subscription -- service-account KEY.json
```

Note: By default, the Instance Validator runs with telemetry validation mode disabled.

Back

Next

Google

Telemetry validation with the Instance Validator

Parameters to enable telemetry validation mode

To enable the mode, run **`instance_validator.py`** with the following parameters:

`-- subscription`
Points to a fully-qualified path to a Google Cloud Pub/Sub subscription (e.g., projects/google.com:your-project/subscriptions/your-subscription).

`-- service-account`
Points to a path to a service account key JSON file corresponding to an account that has permission to pull messages from the Pub/Sub.

Failure to provide both of these parameters will result in early termination of the validator and an error message.

Additional parameters

The following parameters can also be used when telemetry validation mode is enabled:

`-- timeout`
Specifies a timeout duration in seconds for the telemetry validation to run. The default is 600 seconds or 10 minutes.

`-- report-filename`
Provides the validation report in a separate report file.

Telemetry validation with the Instance Validator

What's telemetry validation?

Telemetry validation listens for telemetry messages to validate them against the instance configuration. We recommend performing telemetry validation before completing your building configuration file.

What's needed to perform telemetry validation with the Instance Validator?

In addition to the basic requirements to run the Instance Validator described earlier in this lesson, you'll need the following for telemetry validation:

- A service account key JSON file (point to this file when running telemetry validation)
- A subscription parameter (a path to a Google Cloud subscription)

Back

*Click **Next** to learn how to get a subscription parameter.*

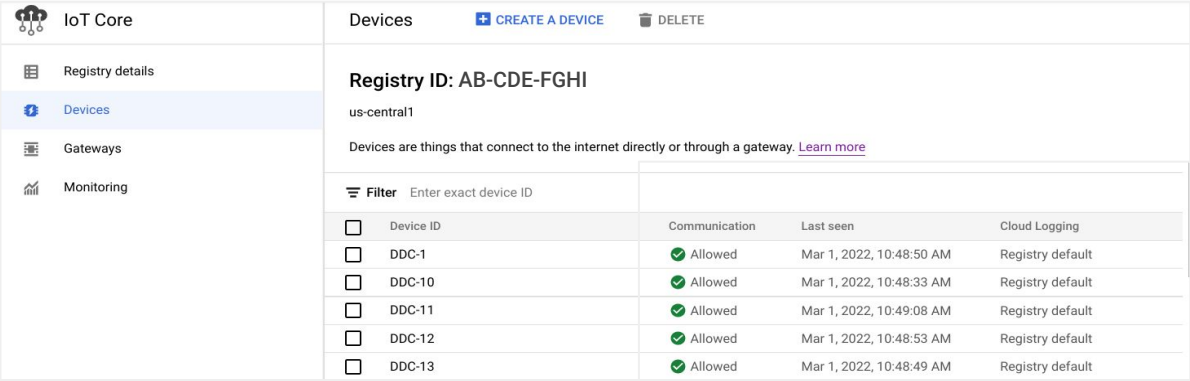
Next

Subscription parameter

Here's how to get a subscription parameter for telemetry validation.

1 2 3 4 5

First, go to Google Cloud Console and select your projects.



Back

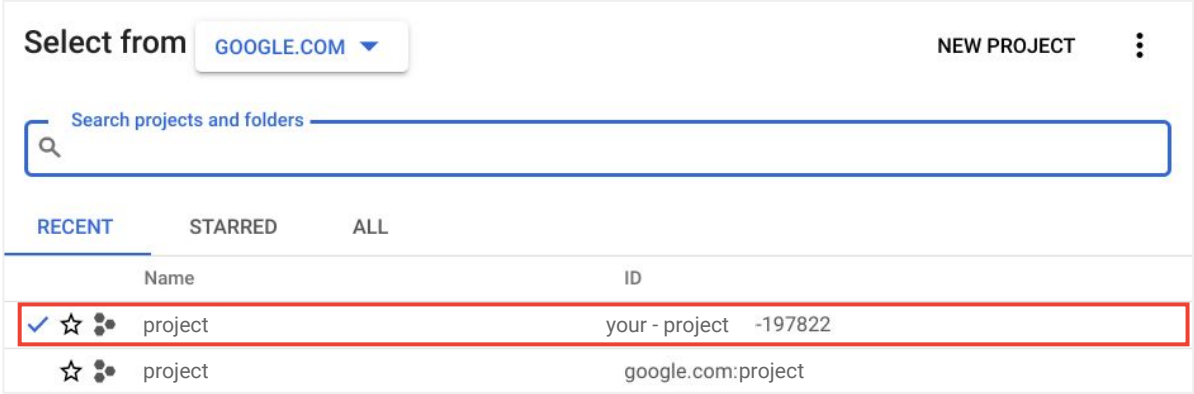
Next

Subscription parameter

Here's how to get a subscription parameter for telemetry validation.

1 2 3 4 5

Navigate to your project.



Back

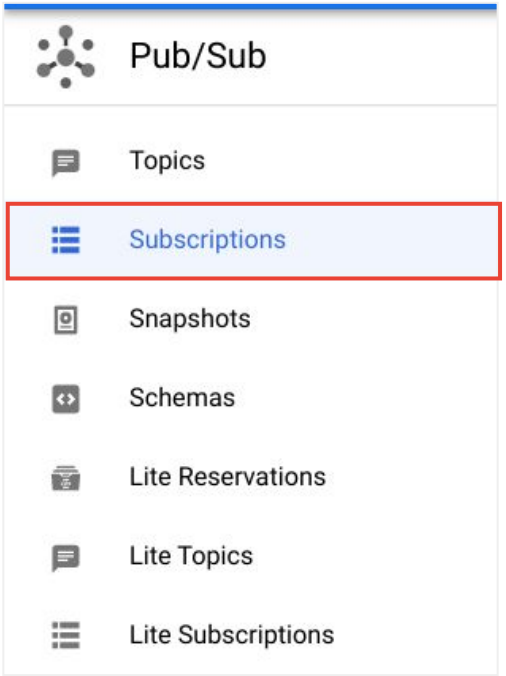
Next

Subscription parameter

Here's how to get a subscription parameter for telemetry validation.

1 2 3 4 5

Navigate to the Pub/Sub portal and select **Subscriptions.**



Back

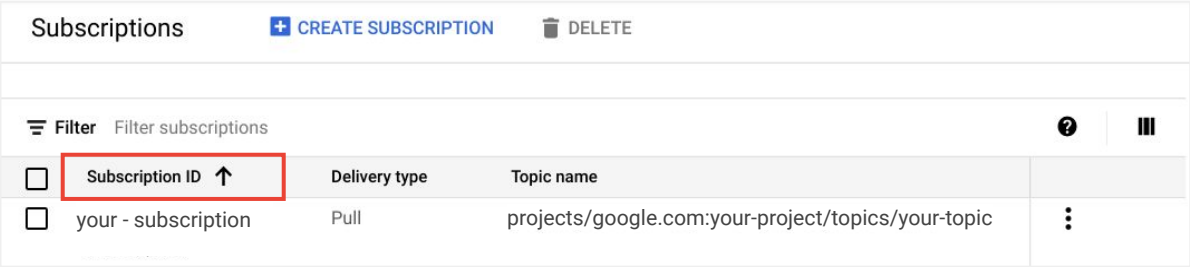
Next

Subscription parameter

Here's how to get a subscription parameter for telemetry validation.

1 2 3 4 5

Filter on the Subscription ID for your project.



Back

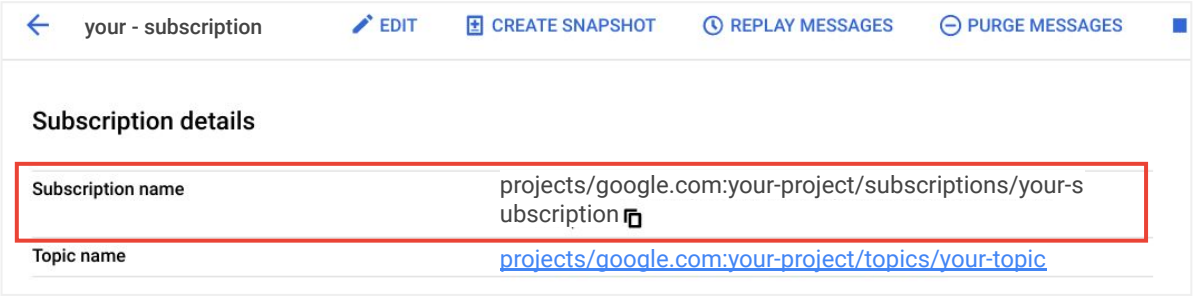
Next

Subscription parameter

Here's how to get a subscription parameter for telemetry validation.

1 2 3 4 5

Select your project to open the Subscription details page.



Here, you'll find the path that forms your subscription parameter:

projects/google.com:your-project/subscriptions/your-subscription

Back

Next

Telemetry validation walkthrough

Let's see how you'd validate the telemetry of a building config file.

1 2 3

First, validate the instance before validating the telemetry.

Run Instance Validator using the following command:

```
python instance_validator.py --input
path/to/YOUR_BUILDING_CONFIG.yaml
```

Make sure the `--input` parameter points to the correct path to the sample building config file.

With the instance validated, you can proceed to perform telemetry validation.

Terminal

Command Prompt

```
C:\[redacted]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python
instance_validator.py --input C:\[redacted]\Desktop\OnboardDemo\demo.yaml

Starting validator...


Starting universe generation...


Starting config validation...


Loading config files...

Validating syntax please wait ...
Opening file: C:\[redacted]\Desktop\OnboardDemo\demo.yaml, please wait ...

Starting config validation...

Validating entities ...
All entities validated
```

Back

Note: You won't be able to practice telemetry validation now. Keep this walkthrough handy to use as a reference when you need to validate telemetry in the near future for your project.

Next

Telemetry validation walkthrough

Let's see how you'd validate the telemetry of a building config file.

1 2 3

Then, proceed with telemetry validation.

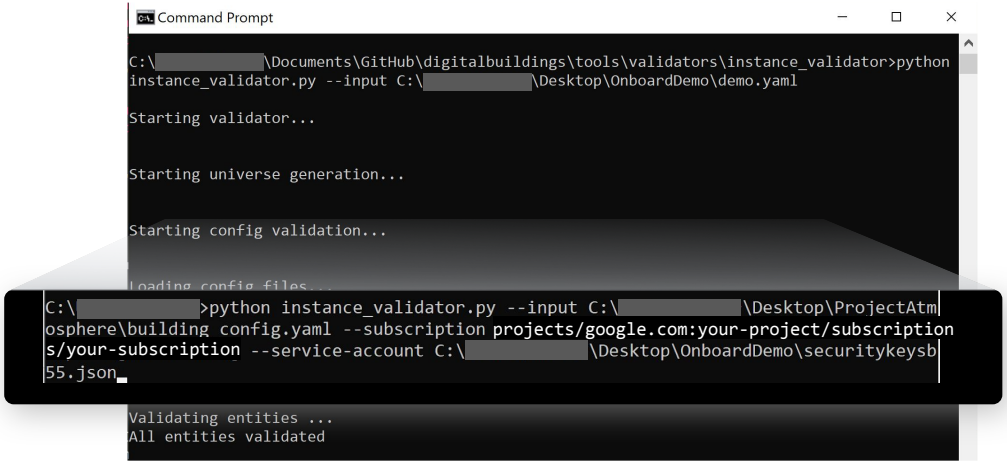
Run Instance Validator with telemetry validation mode enabled using the following command:

```
python instance_validator.py --input path/to/YOUR_BUILDING_CONFIG.yaml -- subscription projects/google.com:your-project/subscriptions/your-subscription -- service-account KEY.json
```

Make sure the `--input` parameter points to the correct path to the sample building config file.

Make sure the `-- subscription` parameter points to a fully-qualified path to a Google Cloud Pub/Sub subscription.

Make sure the `-- service-account` parameter points to the correct path to the service account key file.



Back

Note: You won't be able to practice telemetry validation now. Keep this walkthrough handy to use as a reference when you need to validate telemetry in the near future for your project.

Next

Telemetry validation walkthrough

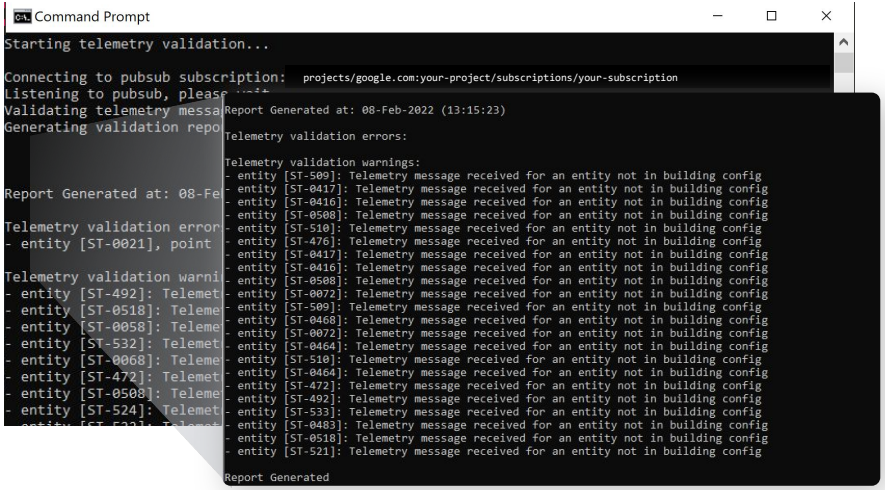
Let's see how you'd validate the telemetry of a building config file.

1 2 3

Finally, review the telemetry validation results.

Here's what it looks like after validation. This was a successful validation with no errors, which is indicated by the blank **`Telemetry validation errors:`** section.

There were some warnings generated, which are listed in the **`Telemetry validation warnings:`** section. These indicate the validated building config doesn't cover every device being published to the topic. This may be intentional, because the building config may be partially complete. However, by the end of the project, it's expected that everything published to the topic is described in the building config, resulting in no warnings after telemetry validation.



Back

Note: You won't be able to practice telemetry validation now. Keep this walkthrough handy to use as a reference when you need to validate telemetry in the near future for your project.

Next

Telemetry validation feedback

After validating the telemetry, the validation results will list errors and warnings for you to address.

Telemetry validation errors

An error is a problem you can't ignore. Errors indicate the telemetry validation failed for some reason, and it must be addressed before validation can continue. All errors must be addressed. Some examples of errors include syntax errors, missing connections, and undefined fields.

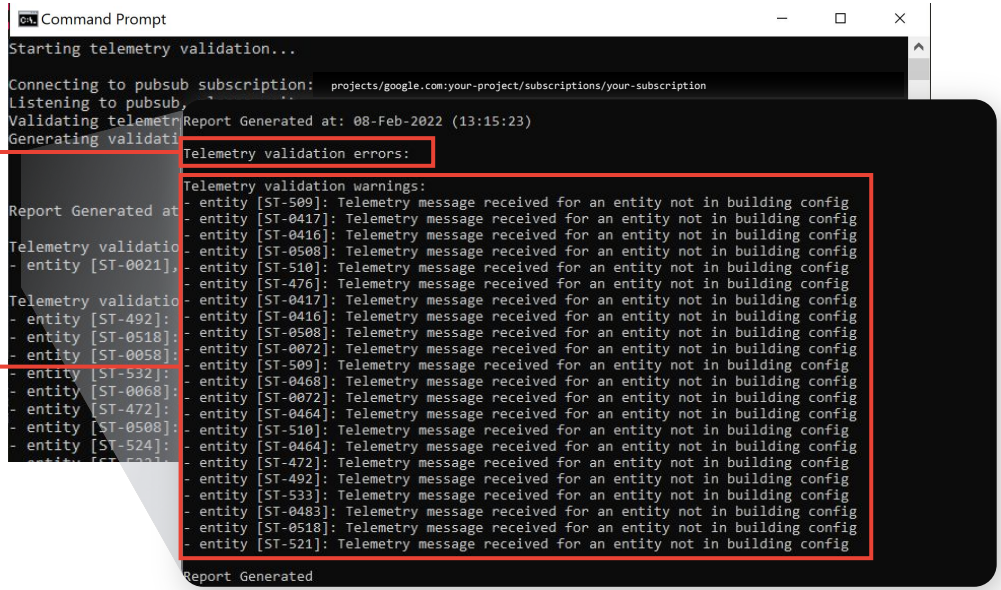
Telemetry validation warnings

A warning is a potential problem that should be assessed. Warnings don't indicate a failed validation, so they don't need to be addressed immediately.

Warnings depend on context, so pay attention to them but don't feel compelled to address them if the current context deems them irrelevant.

For example:

- Ignore warnings if you intend to validate an incomplete building config to test translations you're defining for a few devices.
- Address warnings if you intend to validate a finalized building config.



Back

Next

Telemetry validation feedback (continued)

Let's explore a common telemetry validation error: an invalid path to the payload.

What happens when the path to the payload isn't qualified properly?

As you know by now, we use the `.` in building configs to delimit layers in the UDMI-formatted JSON payload. This format qualifies the path in the payload that contains the value of a field.

Payload

```
points
  raw_field_name
    present_value
```

Building config format

```
points.raw_field_name.present_value
```

Example

Here's a sample payload showing the standard UDMI format for each data point.

```
{
  "timestamp":
  "2022-02-08T16:16:52.000Z",
  "points":
  {
    "co_avg_ppb":
    {
      "present_value": 1024
    },
    "rht_temp_celsius":
    {
      "present_value": 22.31
    },
```

Here's a building config that attempts to translate the point **`co_avg_ppb`** to the field **`thirtysecondrolling_average_zone_air_co_concentration_sensor`**.

Do you see the error?

```
12345678:
  code: ST-0021
  cloud_device_id: "2769931014860840"
  translation:
    thirtysecondrolling_average_zone_air_co_concentration_sensor:
      present_value: co_avg_ppb.present_value
      units:
        key: points.co_avg_ppb.units
        values:
          parts_per_billion: parts_per_billion
```

Current path:

```
present_value: co_avg_ppb.present_value
```

Valid path:

```
present_value: points.co_avg_ppb.present_value
```

Back

*Click **Next** to continue this example.*

Next

Telemetry validation feedback (continued)

Let's explore a common telemetry validation error: an invalid path to the payload.

Example (continued)

After validating the telemetry with the Instance Validator, we see that it's unable to confirm the field is present in the payload.

Here's the mistake and how to fix it

Error

`Entity [ST-0021], point [co_avg_ppb.present_value]: Field missing from telemetry message`

Root cause

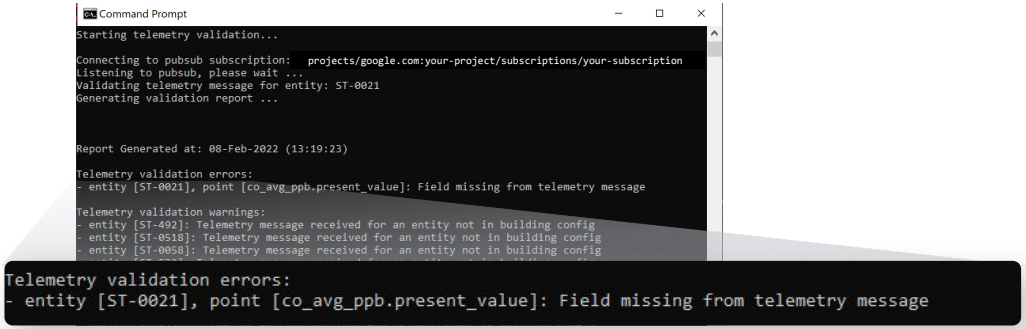
There is an invalid path to the payload in the building config:

`present_value: co_avg_ppb.present_value`

Solution

You'll need to define a valid path in the building config:

`present_value: points.co_avg_ppb.present_value`



Back

Next

Repeat as needed

Instance and telemetry validation should be performed iteratively while you're constructing a building config file and when you're finalizing a building config.

Click on each item to review the commands and parameters.

Run the Instance Validator

Enable telemetry validation

Additional parameters

Back

Next

Repeat as needed

Instance and telemetry validation should be performed iteratively while you're constructing a building config file and when you're finalizing a building config.

Click on each item to review the commands and parameters.

- Run the Instance Validator
- Enable telemetry validation
- Additional parameters

Command to run the Instance Validator

From your terminal or command prompt, use the following command:

```
python instance_validator.py --input path/to/YOUR_BUILDING_CONFIG.yaml
```

Parameters

- `--input` should point to the correct path to the building config file.

Back

Next

Repeat as needed

Instance and telemetry validation should be performed iteratively while you're constructing a building config file and when you're finalizing a building config.

Click on each item to review the commands and parameters.

- Run the Instance Validator
- **Enable telemetry validation**
- Additional parameters

Command to enable telemetry validation

From your terminal or command prompt, use the following command:

```
python instance_validator.py --input path/to/YOUR_BUILDING_CONFIG.yaml -- subscription projects/google.com:your-project/subscriptions/your-subscription -- service-account KEY.json
```

Parameters

- `--input` should point to the correct path to the building config file.
- `-- subscription` should point to a fully-qualified path to a Google Cloud Pub/Sub subscription.
  Example: projects/google.com:your-project/subscriptions/your-subscription
- `-- service-account` should point to the correct path to the service account key JSON file.

Back

Next

Repeat as needed

Instance and telemetry validation should be performed iteratively while you're constructing a building config file and when you're finalizing a building config.

Click on each item to review the commands and parameters.

Run the Instance Validator

Enable telemetry validation

Additional parameters

Additional parameters

Multiple `-- input path/to/YOUR_BUILDING_CONFIG.yaml`
Validates multiple building configs at the same time. You can use as many `--input` parameters for as many building configs you'd like to validate at one time.

`-- report-filename path/to/REPORT-NAME.txt`
Provides the validation results in a separate report file. You'll need to specify your desired location and name for the report file. Report files are needed if you want to share results with another person.

`--modified-ontology-types path/to/modified/ontology/types/folder`
Validates the building config against a local version of a modified DBO. This is handy if you've extended the ontology by adding new modeling concepts to your local ontology but haven't yet submitted them to the DBO or your PR is not yet merged.

`-- timeout`
With telemetry mode enabled, specifies a timeout duration in seconds for the telemetry validation to run.

Back

Next

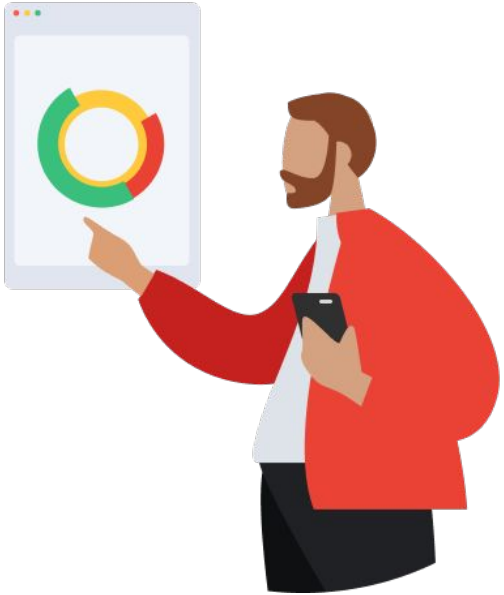
Lesson 7 summary

Let's review what you learned about:

- The Instance Validator tool
- Instance validation
- Telemetry validation
- Validation feedback

Now you should be able to:

- Run the Instance Validator with basic commands and additional parameters.
- Generate a report file to share validation results with others.
- Identify the root cause of validation errors and correct them.
- Enable telemetry validation mode in the Instance Validator.



Back

Next

Workflow revisited

Here's one last look at the recommended workflow for data modeling from Lesson 1.

Remember, data modeling with the DBO heavily depends on your ability to accurately identify the equipment that needs to be modeled and how that equipment can be described using the DBO. Use our recommended workflow along with the resources in the Digital Buildings Project GitHub repo to guide your efforts.

1. Determine which devices need to be modeled
2. Determine which data points are required
3. Name each data point using the DBO
4. Propose an ontology extension
5. Construct and finalize the building configuration file
6. Validate the instance and telemetry
7.

Complete!

After you've validated the building config file with the Instance Validator, consider it complete and ready to move forward in your project.

Back

Next

You completed Lesson 7!

This also completes Module 2: Data modeling with the DBO.

Back

Helpful resources

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

- Instance Validator
  Used to validate a building config file to ensure it conforms to the DBO.
- Ontology Explorer
  Used to ask basic questions of what's curated within the DBO.
- Digital Buildings Project GitHub
  Contains source code, tooling, and documentation for the DBO.