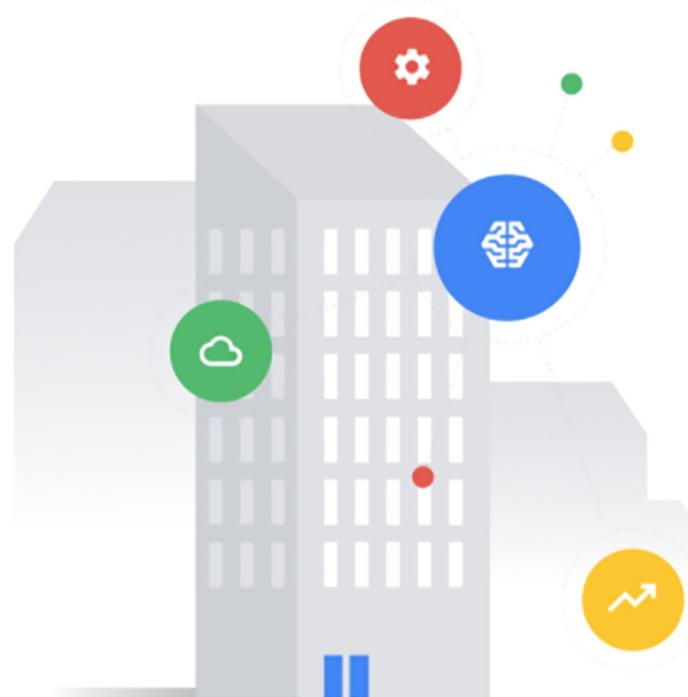




Module 2 | Lesson 7



# 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.

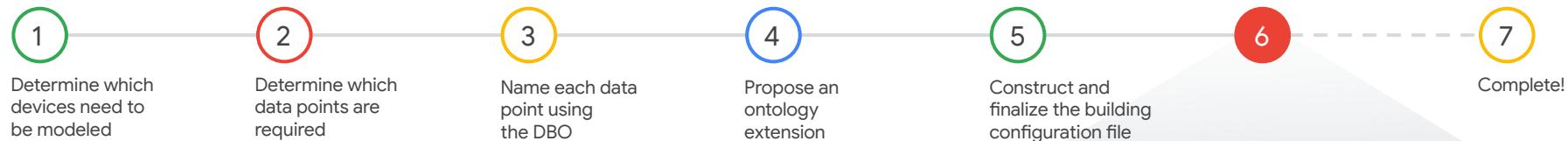
Ready to get started?

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.



## 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:

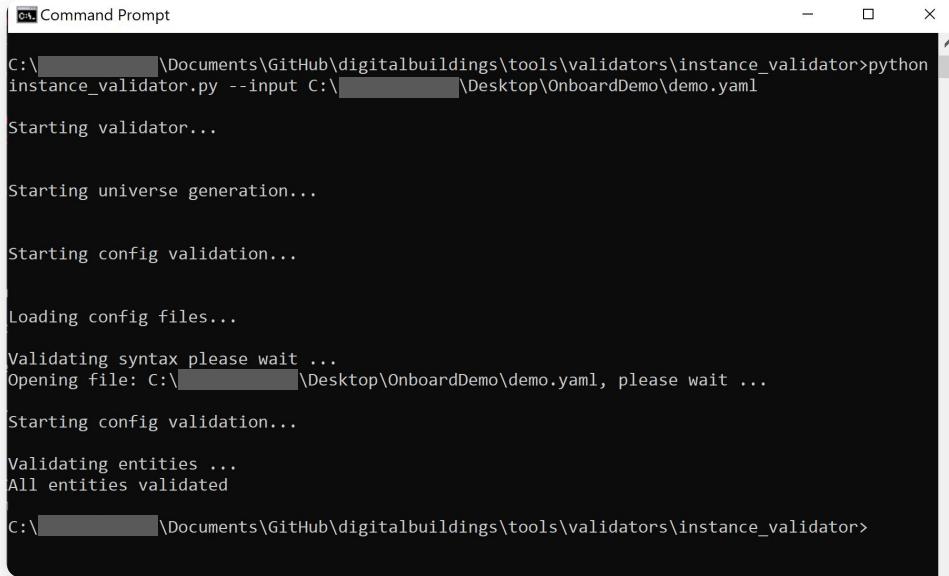
- Run the Instance Validator with basic commands and additional parameters.
- Generate a report file to share validation results with others.
- Identify the root cause of validation errors and correct them.
- Enable telemetry validation mode in the Instance Validator.

[Back](#)[Next](#)



# 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](#)

# 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).

[Back](#)

[Next](#)

# Instance Validator (continued)

## What's needed to run the Instance Validator?

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

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

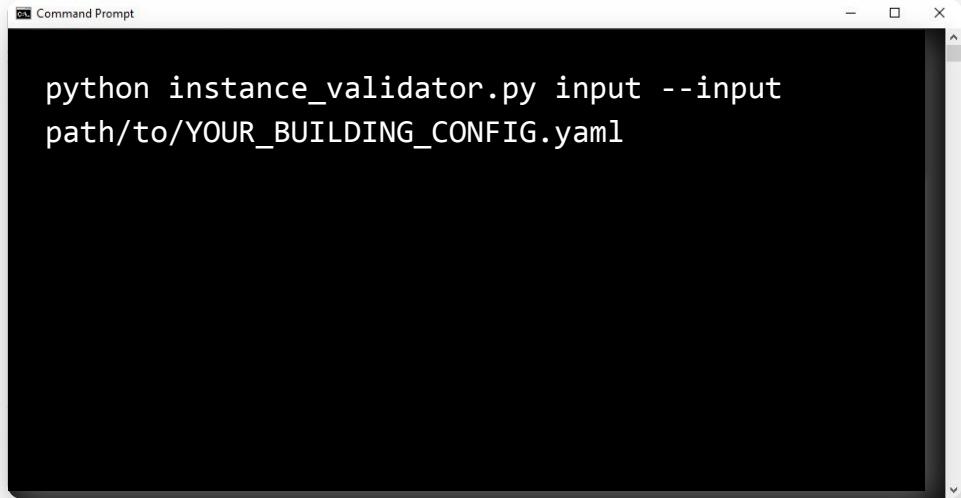
[Back](#)

[Next](#)



# Instance validation with the Instance Validator

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

[Back](#)[Google](#)

```
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.

[Next](#)

# 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.

[Back](#)

Google

*Click [Next](#) when you're ready to begin.*

[Next](#)

# 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.

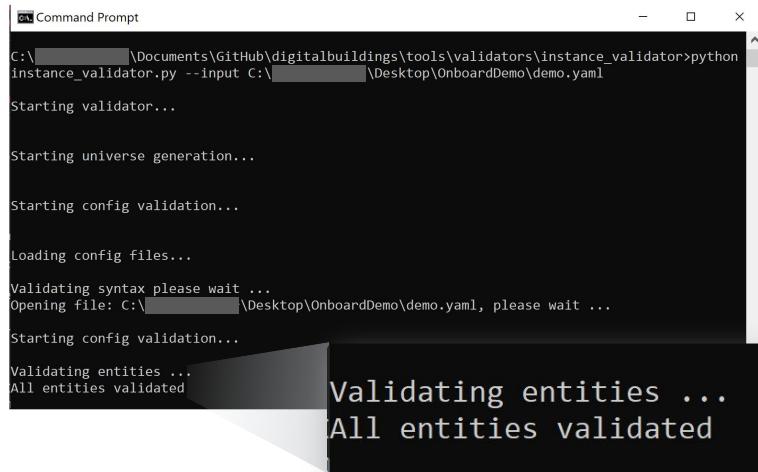
[Next](#)

# Practice 1

## Check your work!

The validation results should have been successful.

**Did you end up with this success message?**



```
C:\> Command Prompt  
C:\> cd Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python  
instance_validator.py --input C:\> Desktop\OnboardDemo\demo.yaml  
  
Starting validator...  
  
Starting universe generation...  
  
Starting config validation...  
  
Loading config files...  
  
Validating syntax please wait ...  
Opening file: C:\> Desktop\OnboardDemo\demo.yaml, please wait ...  
  
Starting config validation...  
  
Validating entities ...  
All entities validated
```

Validating entities ...  
All entities validated

[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



Back

Google

**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.

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

Next

# 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.

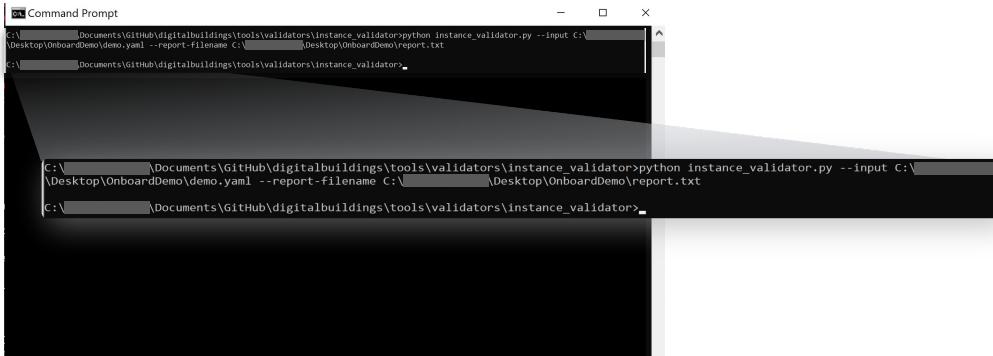
[Next](#)

# 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 report file showing the validation results..

```
1  Starting validator...
2
3
4
5  Starting universe generation...
6
7
8  Starting config validation...
9
10
11  Loading config files...
12
13  Validating syntax please wait ...
14  Opening file: C:\Users\[REDACTED]\OnboardDemo\demo.yaml, please wait ...
15
16  Starting config validation...
17
18  Validating entities ...
19  All entities validated
20
```

[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.

```
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.

```
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 ...
File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator.py", line 10, in <module>
    config = _ValidateConfig(filenames, universe)
          ^
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](#)

[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

### 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\_bad\_field.yaml

demo\_bad\_indent.yaml

demo\_bad\_state.yaml

demo\_missing\_building.yaml

demo\_missing\_keys.yaml

demo\_missing\_type.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?**

### Terminal

```
C:\Users\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python instance_validator.py validate demo_bad_connection.yaml
Starting config validation...
Starting validating entities...
Validating entities ...
Orphan connection to: US-SVL-MP2-2-FAKEZONE
EF-1 is not a valid instance
All entities validated
```

[Back](#)

Click [Next](#) for info about the error, root cause, and solution.

[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](#)

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

[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](#)

Google

## 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?**

### Terminal

```
C:\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python
Instance_validator
Starting validation...
Starting universal validation...
Validating entities ...
Field banana_sensor is not defined on the type
Required field /run_command is missing from translation
EF-1 is not a valid instance
All entities validated

Starting config validation...
Validating entities ...
Opening file: C:\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator\demo_bad_field.yaml
Starting config validation...
Validating entities ...
```

Click [Next](#) for info about the error, root cause, and solution.

[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](#)

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

[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

Click **Next** for info about the error, root cause, and solution.

Next

## 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?**

### Terminal

```
C:\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python instance_validator.py --input C:\Users\Trevor\Desktop\OnboardDemo\demo.yaml
Starting validation...
Starting config validation...
Starting universe...
Loading config files...
Starting config validation...
Validating syntax please wait ...
Opening file: C:\Users\Trevor\Desktop\OnboardDemo\demo_bad_indent.yaml, please wait ...
Mapping values are not allowed here
in "unicode string", line 7, column 7:
      id: 12345
      ^ (line: 7)
Starting config validation...
Validating entities ...
```

# 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](#)

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

## Check your work!



Here's the mistake and how to fix it

### Error

The validation message specifies exactly where the error is located:

in "<unicode string>", line 7 column 7:

id: 12345

### Root cause

This requires checking the building config to find the root cause: the indent is incorrect due to extra spaces next to id: 12345.

### Solution

You'll need to delete the extra spaces before id: 12345.

### Building config

```
5
6   EF-1:
7     type: HVAC/FAN_SS
8     id: 12345
9     guid: 12345678
10    cloud_device_id: 1234567
11    translation:
12      run_command:
13        present_value: "points.fan_ss.present_value"
14        states:
15          ON: "true"
16          OFF: "false"
17      run_status:
18        present_value: "points.fan_sts.present_value"
19        states:
20          OFF: "false"
21          ON: "true"
```

[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](#)

Click [Next](#) for info about the error, root cause, and solution.

[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](#)

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

[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\_bad\_field.yaml

demo\_bad\_indent.yaml

demo\_bad\_state.yaml

demo\_missing\_building.yaml

demo\_missing\_keys.yaml

demo\_missing\_type.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?**

### Terminal

```
Command Prompt
Starting config validation...
Validating entities ...
Config must contain a non-deleted entity with a building type
Starting validator...
Traceback (most recent call last):
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator.py", line 25, in <module>
    instance_validator.main()
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator.py", line 16, in main
    validate_config(config)
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator.py", line 10, in validate_config
    validate_config_file(config)
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\validator.py", line 20, in validate_config_file
    validate_entities(entities)
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\validator.py", line 14, in validate_entities
    validate_entity(entities)
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\validator.py", line 22, in validate_entity
    validate_type(entity)
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\validator.py", line 25, in validate_type
    validate_type_error(entity)
  File "C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\validator.py", line 14, in validate_type_error
    raise SyntaxError('Building Config must contain an entity with a building type')
SyntaxError: Building Config must contain an entity with a building type
```

[Back](#)

Click [Next](#) for info about the error, root cause, and solution.

[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](#)

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

[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

Click **Next** for info about the error, root cause, and solution.

Next

## 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?**

### Terminal

Command Prompt

```
C:\...\Documents\GitHub\instance_validator.py --input C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
Starting validator...  
Starting universe generation...  
Starting config validation...  
Loading config files...  
Validating syntax please wait ...  
Opening file: C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
entities = ValidateConfig(filenames, universe)  
File "C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
entities, config_mode = Deserialize(filenames)  
File "C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
parser.Finalize()  
File "C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
ize  
self.ProcessEntities()  
File "C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
cessEntities  
self._ValidateBlock(self._queued_entity_blocks.popLeft(),  
File "C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
ateBlock  
validation_fn(validated)  
File "C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
dateEntityBlock  
self._ValidateEntityContent(block.get(key))  
File "C:\...\Documents\GitHub\digitalbuildings\tools\validators\d  
ateEntityContent  
raise KeyError('cloud_device_id required when translation is present.')  
KeyError: 'cloud_device_id required when translation is present.'
```

# 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](#)

*Click [Next](#) for info about the second error.*

[Next](#)

## Check your work!



Here's the mistake and how to fix it

### Error

```
KeyError: 'cloud_device_id required when translation is present.'
```

### Root cause

A translation is missing `cloud_device_id`.

### Solution

You'll need to add `cloud_device_id` to the affected translation.

## Wait!

There's actually another error present in this building config. Apply the above solution, and then try validating the file again.

# 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



After fixing the first error and validating the building config again, you'll see another error message indicating `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 validate in small batches to avoid a lot of hidden errors in subsequent validation runs.

```
Terminal

Command Prompt
C:\...\Documents\Gi
instance_validator.py --input
Starting validator...
Starting universe generation
Starting config validation.
Starting config validation...
Loading config files...
Loading config files...
Validating syntax please wait ...
Opening file: C:\...\Desktop\OnboardDemo\demo_missing_keys
[WARNING]: Entity GUID will be required in the future for None.

Starting config validation...
Starting config validation...
Starting config validation...
Starting config validation...
Validating entities ...
All entities require IDs
WARNING: Entity GUID will be required in the future.
EF-1 is not a valid instance
All entities validated
Validating entities ...
```

Back

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

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](#)

Click [Next](#) for info about the error, root cause, and solution.

[Next](#)

## 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?**

Terminal

```
C:\> cd 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:\> while parsing a mapping in "<unicode string>", line 5, column 1:>   EF-1:>     ^ (line: 5)> required key(s) 'type' not found in "<unicode string>", line 19, column 1:>           ON: 'true'>             ^ (line: 19)
```

# 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](#)

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

[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\_value.yaml" should have resulted in an error. **Did you end up with this error message?**

### Terminal

```
Command Prompt
C:\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator.py --input C:\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator.py
Starting validator...
Starting universe generation...
Starting config validation...
Loading config files...
Validating syntax please wait ...
Opening file: C:\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator.py
Starting config validation...
Validating entities ...
raise ValueError('raw_field_name cannot be empty')
ValueError: raw_field_name cannot be empty
```

Back

Click **Next** for info about the error, root cause, and solution.

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](#)

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

Google

## Check your work!



Here's the mistake and how to fix it

### Error

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

### Root cause

This requires checking the building config to find the root cause: there's a missing value for the key `present_value`.

### Solution

You'll need to qualify the corresponding point from the JSON payload.

**Building config**

```
6 ▼ EF-1:
7   type: HVAC/FAN_SS
8   id: 12345
9   guid: 12345566
10  cloud_device_id: 1234567
11  translation:
12    run_command:
13      present_value: [REDACTED]
14    states:
15      ON: "true"
16      OFF: "false"
17  run_status:
18    present_value: "points.fan_sts.present_value"
19    states:
20      OFF: "false"
21      ON: "true"
```

[Next](#)

## Lesson 7

# Practice 4



[Back](#)

Google

**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.*

[Next](#)

# 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

```
Command Prompt
C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python
instance_validator.py --input C:\Users\Trevor\Desktop\OnboardDemo\demo.yaml

Starting validator...
Starting universe generation...
Starting config validation...
Loading config files...
Validating syntax please wait ...
Opening file: C:\[REDACTED]\OnboardDemo\demo.yaml
Starting config validation...
Validating entities ...
Field supply_fan_speed_frequency_sensor is not defined on the type
EF-1 is not a valid instance
All entities validated
```

[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

```
Command Prompt
C:\...\Documents\GitHub\digitalbuildings\tools\validators\instance_validator>python
instance_validator.py --input C:\Users\Trevor\Desktop\OnboardDemo\demo.yaml
Starting validator...
Starting universe generation...
Starting config validation...
Loading config files...
Validating syntax please wait...
Opening file: C:\...
Starting config validation...
Validating entities ...
All entities validated
```

### What was the error?

The original building config file had a field in the entity **EF-1**'s translation 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](#)

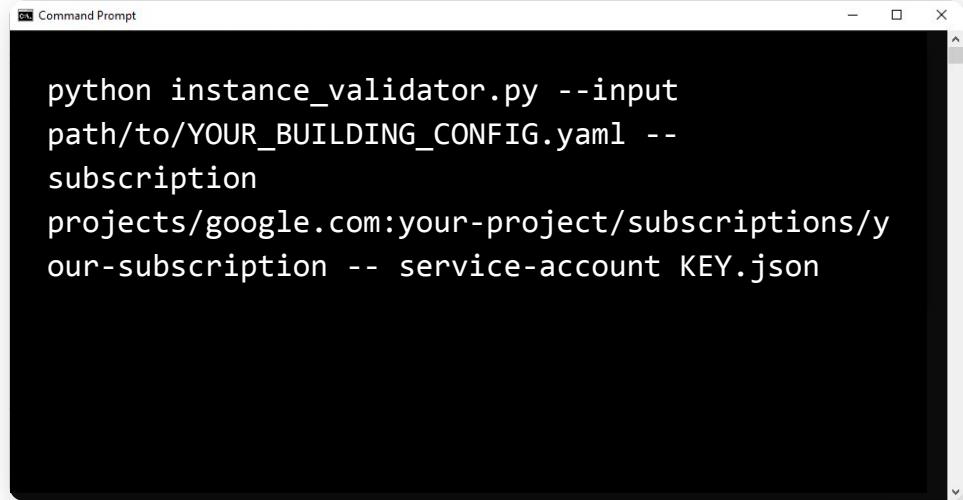
Click **Next** to move on to telemetry validation.

[Next](#)



# Telemetry validation with the Instance Validator

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



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

[Back](#)

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

[Next](#)

# 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.

[Back](#)

[Next](#)

# 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.

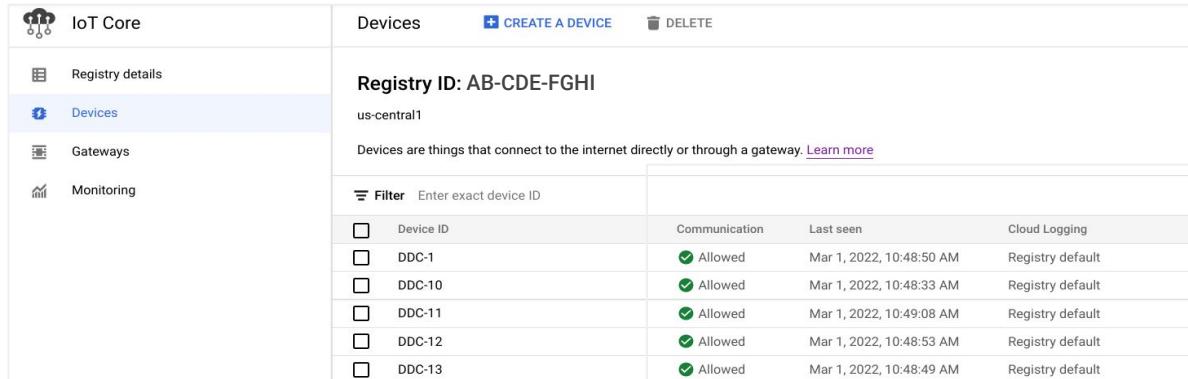
[Next](#)

# 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.



The screenshot shows the Google Cloud IoT Core interface. On the left, there's a sidebar with icons for Registry details, Devices (which is selected), Gateways, and Monitoring. The main area has a header "Devices" with "CREATE A DEVICE" and "DELETE" buttons. Below the header, it says "Registry ID: AB-CDE-FGHI" and "us-central1". A note states "Devices are things that connect to the internet directly or through a gateway." with a "Learn more" link. There's a "Filter" input field. The table lists six devices: DDC-1, DDC-10, DDC-11, DDC-12, and DDC-13. Each row includes a checkbox for "Device ID", "Communication" status (all show "Allowed" with a green checkmark), "Last seen" timestamp (all show "Mar 1, 2022, 10:48:50 AM"), and "Cloud Logging" status (all show "Registry default").

Device ID	Communication	Last seen	Cloud Logging
DDC-1	Allowed	Mar 1, 2022, 10:48:50 AM	Registry default
DDC-10	Allowed	Mar 1, 2022, 10:48:33 AM	Registry default
DDC-11	Allowed	Mar 1, 2022, 10:49:08 AM	Registry default
DDC-12	Allowed	Mar 1, 2022, 10:48:53 AM	Registry default
DDC-13	Allowed	Mar 1, 2022, 10:48:49 AM	Registry default

Back

Next

# Subscription parameter

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

- 1
- 2
- 3
- 4
- 5

Navigate to your project.

The screenshot shows a search interface for selecting a project. At the top, it says "Select from GOOGLE.COM" with a dropdown arrow. To the right are buttons for "NEW PROJECT" and three vertical dots. Below is a search bar with the placeholder "Search projects and folders". Underneath are three tabs: "RECENT" (underlined), "STARRED", and "ALL". A table follows, with columns for "Name" and "ID". The first row has a checked checkbox, a star icon, a copy icon, and the name "project". To its right are the ID "your - project -197822" and a link "google.com:project". The second row has an unchecked checkbox, a star icon, a copy icon, and the name "project".

Name	ID
✓ ★ ⚡ project	your - project -197822
★ ⚡ project	google.com:project

[Back](#)

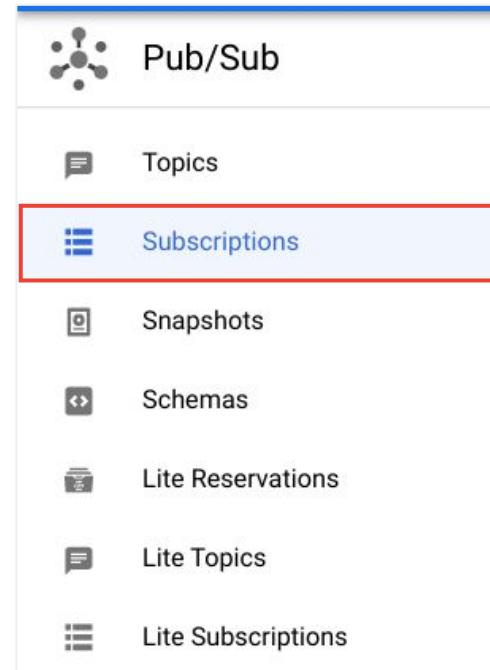
[Next](#)

# 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.

Subscriptions		
<a href="#">+ CREATE SUBSCRIPTION</a> <a href="#">DELETE</a>		
<a href="#">Filter</a> Filter subscriptions		
<input type="checkbox"/> Subscription ID ↑	Delivery type	Topic name
<input type="checkbox"/> your - subscription	Pull	projects/google.com:your-project/topics/your-topic

[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.

The screenshot shows the 'Subscription details' page for a subscription named 'your - subscription'. The 'Subscription name' field contains the value 'projects/google.com:your-project/subscriptions/your-subscription'. The 'Topic name' field contains the value 'projects/google.com:your-project/topics/your-topic'. Both fields are highlighted with a red border. The page also includes standard navigation and action buttons like 'EDIT', 'CREATE SNAPSHOT', 'REPLAY MESSAGES', and 'PURGE MESSAGES'.

Subscription details	
Subscription name	projects/google.com:your-project/subscriptions/your-subscription
Topic name	<a href="https://cloud.google.com:your-project/topics/your-topic">projects/google.com:your-project/topics/your-topic</a>

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

[projects/google.com:your-project/subscriptions/your-subscription](https://cloud.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

```
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.

```
C:\[REDACTED]\Documents\GitHub\digitalbuildings\tools\validators>python instance_validator.py --input C:\[REDACTED]\Desktop\OnboardDemo\demo.yaml  
Starting validator...  
Starting universe generation...  
Starting config validation...  
Loading config files...  
C:\[REDACTED]>python instance_validator.py --input C:\[REDACTED]\Desktop\ProjectAtmosphere\building_config.yaml --subscription projects/google.com:your-project/subscriptions/your-subscription --service-account C:\[REDACTED]\Desktop\OnboardDemo\securitykeysb55.json  
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

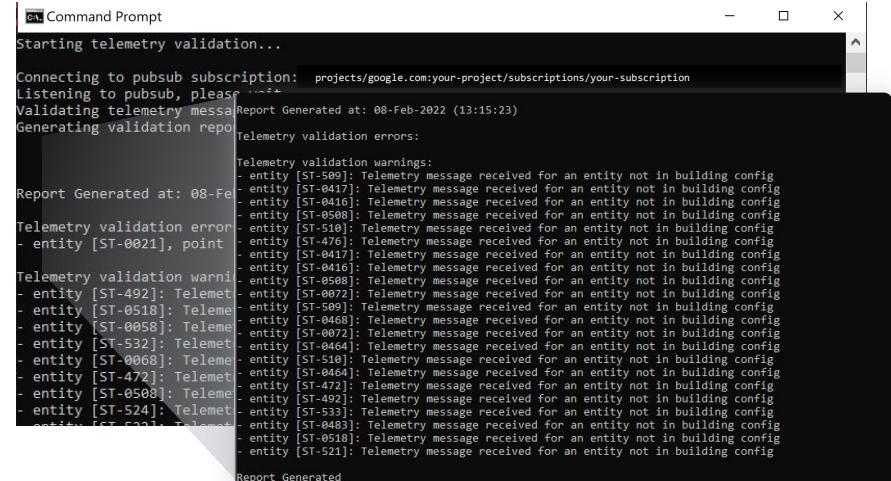
Let's see how you'd validate the telemetry of a building config file.

- 1
- 2
- 3

## 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.



The screenshot shows a Command Prompt window titled "Command Prompt". The output of the command "Starting telemetry validation..." is displayed. It includes a report header with the date "Report Generated at: 08-Feb-2022 (13:15:23)". The validation process connects to a pubsub subscription and listens for messages. It then validates the telemetry messages and generates a validation report. The validation report lists "Telemetry validation errors:" and "Telemetry validation warnings:". The errors section is empty. The warnings section contains numerous entries, each detailing a telemetry message received for an entity not in the building config, such as "entity [ST-509]: Telemetry message received for an entity not in building config".

```
Starting telemetry validation...
Connecting to pubsub subscription: projects/google.com:your-project/subscriptions/your-subscription
Listening to pubsub, please ...-+
Validating telemetry messa Report Generated at: 08-Feb-2022 (13:15:23)
Generating Validation repo
Telemetry validation errors:
Telemetry validation warnings:
- entity [ST-509]: Telemetry message received for an entity not in building config
- entity [ST-0417]: Telemetry message received for an entity not in building config
- entity [ST-0508]: Telemetry message received for an entity not in building config
- entity [ST-510]: Telemetry message received for an entity not in building config
- entity [ST-0416]: Telemetry message received for an entity not in building config
- entity [ST-0417]: Telemetry message received for an entity not in building config
- entity [ST-0416]: Telemetry message received for an entity not in building config
- entity [ST-0508]: Telemetry message received for an entity not in building config
- entity [ST-0072]: Telemetry message received for an entity not in building config
- entity [ST-492]: Telemetry message received for an entity not in building config
- entity [ST-0518]: Telemetry message received for an entity not in building config
- entity [ST-0058]: Telemetry message received for an entity not in building config
- entity [ST-532]: Telemetry message received for an entity not in building config
- entity [ST-0068]: Telemetry message received for an entity not in building config
- entity [ST-472]: Telemetry message received for an entity not in building config
- entity [ST-0508]: Telemetry message received for an entity not in building config
- entity [ST-524]: Telemetry message received for an entity not in building config
- entity [ST-521]: Telemetry message received for an entity not in building config
- entity [ST-492]: Telemetry message received for an entity not in building config
- entity [ST-0468]: Telemetry message received for an entity not in building config
- entity [ST-0072]: Telemetry message received for an entity not in building config
- entity [ST-0464]: Telemetry message received for an entity not in building config
- entity [ST-510]: Telemetry message received for an entity not in building config
- entity [ST-0464]: Telemetry message received for an entity not in building config
- entity [ST-472]: Telemetry message received for an entity not in building config
- entity [ST-492]: Telemetry message received for an entity not in building config
- entity [ST-533]: Telemetry message received for an entity not in building config
- entity [ST-0483]: Telemetry message received for an entity not in building config
- entity [ST-0518]: Telemetry message received for an entity not in building config
- entity [ST-521]: Telemetry message received for an entity not in building config
Report Generated
```

[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.

```
Command Prompt
Starting telemetry validation...
Connecting to pubsub subscription: projects/google.com:your-project/subscriptions/your-subscription
Listening to pubsub...
Validating telemetry...
Report Generated at: 08-Feb-2022 (13:15:23)
Generating validation report...
Telemetry validation errors:
Report Generated at: 08-Feb-2022 (13:15:23)
Telemetry validation errors:
- entity [ST-0021]: Telemetry message received for an entity not in building config
- entity [ST-00417]: Telemetry message received for an entity not in building config
- entity [ST-00416]: Telemetry message received for an entity not in building config
- entity [ST-00508]: Telemetry message received for an entity not in building config
- entity [ST-510]: Telemetry message received for an entity not in building config
- entity [ST-476]: Telemetry message received for an entity not in building config
- entity [ST-04417]: Telemetry message received for an entity not in building config
- entity [ST-04416]: Telemetry message received for an entity not in building config
- entity [ST-05088]: Telemetry message received for an entity not in building config
- entity [ST-0072]: Telemetry message received for an entity not in building config
- entity [ST-509]: Telemetry message received for an entity not in building config
- entity [ST-0468]: Telemetry message received for an entity not in building config
- entity [ST-0072]: Telemetry message received for an entity not in building config
- entity [ST-0464]: Telemetry message received for an entity not in building config
- entity [ST-510]: Telemetry message received for an entity not in building config
- entity [ST-0464]: Telemetry message received for an entity not in building config
- entity [ST-472]: Telemetry message received for an entity not in building config
- entity [ST-492]: Telemetry message received for an entity not in building config
- entity [ST-533]: Telemetry message received for an entity not in building config
- entity [ST-0483]: Telemetry message received for an entity not in building config
- entity [ST-0518]: Telemetry message received for an entity not in building config
- entity [ST-521]: Telemetry message received for an entity not in 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

```
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?

```
ST-0021:
  cloud_device_id: '2769931014860840'
  guid: '123'
  id: '123'
  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
```

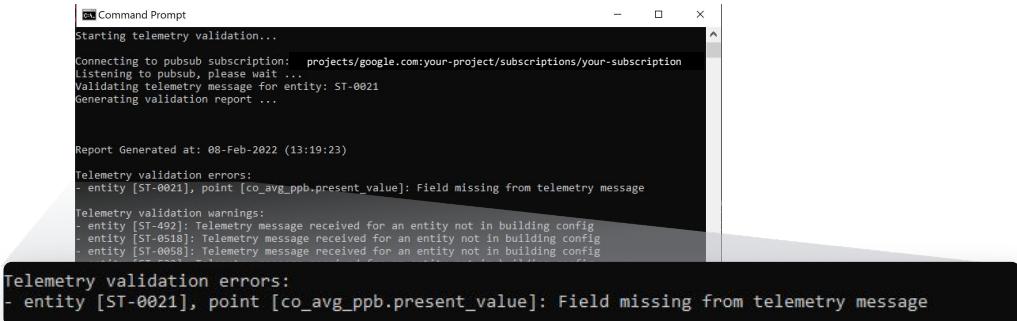
### Root cause

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
```



The screenshot shows a Command Prompt window titled "Command Prompt". It displays the following output:

```
Starting telemetry validation...
Connecting to pubsub, please wait ...
Listening to pubsub, please wait ...
Validating telemetry message for entity: ST-0021
Generating validation report ...

Report Generated at: 08-Feb-2022 (13:19:23)

Telemetry validation errors:
- entity [ST-0021], point [co_avg_ppb.present_value]: Field missing from telemetry message

Telemetry validation warnings:
- entity [ST-492]: Telemetry message received for an entity not in building config
- entity [ST-051B]: Telemetry message received for an entity not in building config
- entity [ST-005B]: Telemetry message received for an entity not in building config
```

A callout bubble points to the error message in the validation report:

Telemetry validation errors:  
- entity [ST-0021], point [co\_avg\_ppb.present\_value]: Field missing from telemetry message

[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:

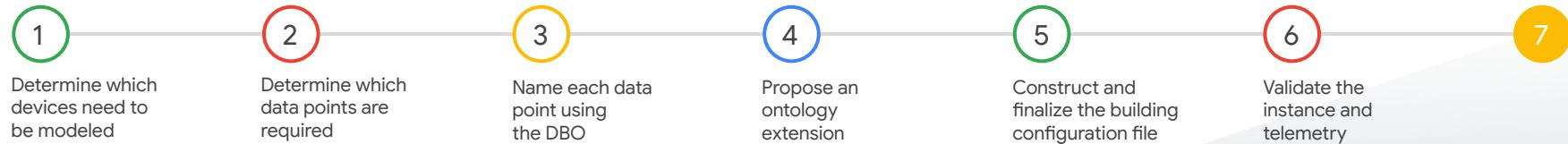
- Run the Instance Validator with basic commands and additional parameters.
- Generate a report file to share validation results with others.
- Identify the root cause of validation errors and correct them.
- Enable telemetry validation mode in the Instance Validator.

[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.



**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](#)

Google

## Helpful resources

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

- [Digital Buildings Project GitHub](#)

Contains source code, tooling, and documentation for the DBO.

- [Instance Validator](#)

Used to validate a building configuration file to ensure it conforms to the DBO.

- [Ontology Explorer](#)

Used to ask basic questions of what's curated within the DBO.