JSON guide for LCAbyg version 5.3.1.0

Creation of JSON projects in LCAbyg - import

LCAbyg
{...} JSON

Title	JSON guide for LCAbyg 2023
Subtitle text	Creation of JSON projects in LCAbyg - import
Published	December 2020
Last update	December 2022
Authors	Terese Pagh, Christian Grau Sørensen, Anne Mathilde Gøtke
Keywords	JSON, JavaScript Object Notation, LCAbyg
Publisher	BUILD - Institut for Byggeri, By og Miljø, Aalborg Universitet, A.C. Meyers Vænge 15, 2450 København SV
Email	lcabyg@build.aau.dk
Website	https://www.lcabyg.dk/da/

Disclaimer

LCAbyg is under constant development and for that reason, there may be changes in the json files. Always remember to download the latest version of LCAbyg and download the latest version of the JSON guide and JSON examples, which you can find here https://www.lcabyg.dk/da/download-legacy/.

You can sign up for the LCAbyg newsletter to receive updates and bug fixes here https://www.lcabyg.dk/newsletter

Frequently asked questions can be found at LCAbyg website here: https://lcabyg.dk/da/faq/

If you cannot find an answer to your questions or if you have feedback, write to the LCAbyg mailbox lcabyg@build.aau.dk.

Readme

Before starting, you should carefully read the instructions in this chapter. In the zip file, containing this JSON guide, you will find several other important folders. E.g. the **import_example** folder contains several JSON files which the user can edit and add to. Furthermore, the zip file also contains a folder named **gen_dk** which contains all the IDs and names from GenDK (the library in LCAbyg), see Figure 1.

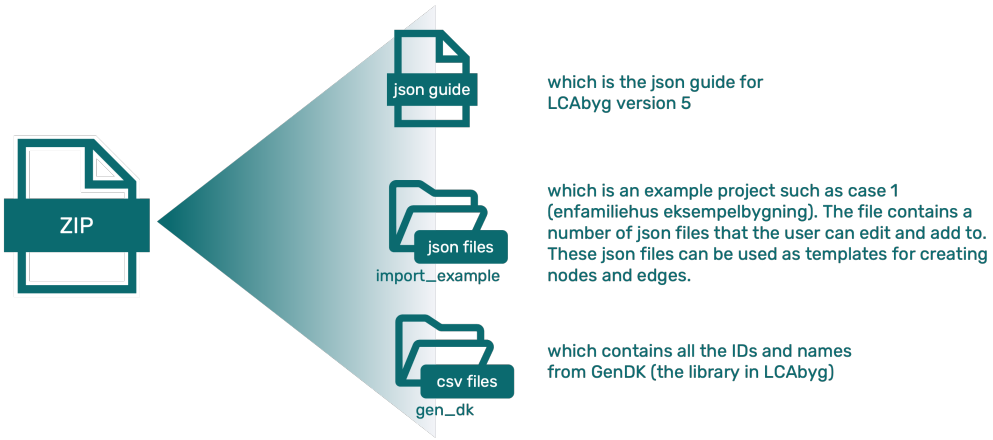


Figure 1: The contents of the zip file

Before starting, you should unzip (extract) the **import_example.zip** file and locate it in the LCAbyg 5.3.1.0 folder. When downloading LCAbyg you chose the location of the LCAbyg 5.3.1.0 folder. In the vast majority of the cases, the folder is located in **C:\Program Files\SBi**.

Chapter 1

Introduction and Guide build up

1.1 Introduction

This JSON guide for LCAbyg version 5 is about the technology used behind the program for quicker import and export in LCAbyg. LCAbyg is a tool used to model a Life Cycle Assessment (LCA) which calculates the building's environmental profile and resource consumption.

In LCAbyg version 3.2 it is possible to create a third party integration i.e. connecting LCAbyg with a program developed by others than BUILD. In the new version of LCAbyg this integration has been reestablished. In LCAbyg 3.2 the xml file format is used for this integration whereas LCAbyg 5.0 the open file format JSON is used.
Before reading the JSON guide for LCAbyg we recommend you to read the newest user guide for LCAbyg. The guide is in Danish and is available at https://www.lcabyg.dk/da/vejledning/brugervejledning-lcabyg-5/ (LCAbyg 5 brugervejledning).

1.1.1 The JSON file format

JSON stands for JavaScript Object Notation and is an open standard for storing and exchanging data. LCAbyg contains several JSON files in which the user, to some extent, can use and modify. E.g., when creating a construction (konstruktion) in LCAbyg, a JSON file is made and placed in the representative folder which is linked to LCAbyg, see Figure 1.1 and 1.2.

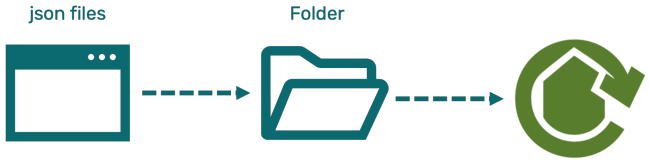


Figure 1.1: Simplification of how the JSON files are linked to LCAbyg

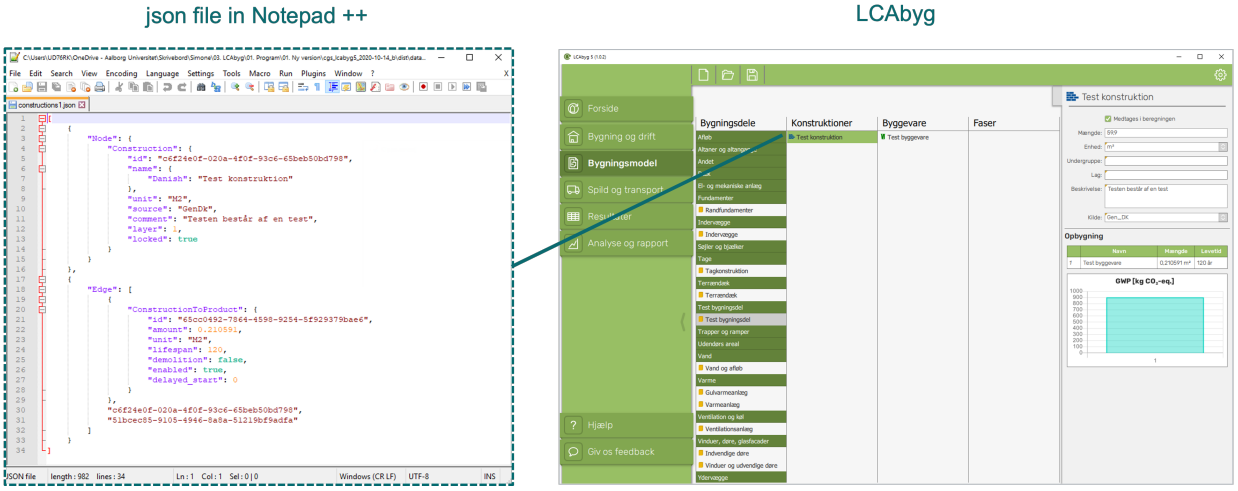


Figure 1.2: The connection between LCAbyg and a JSON file. Left: a new construction is made in a JSON file and connected to a building part. Right: the new construction is visible in LCAbyg

1.1.2 Working in the JSON file format

A project template named *Import Example* is created with the purpose that the user can work and edit in the json files located in the associated folder (named import_example). This guide is based on this project template, and therefore we recommended that you work in this template, to begin with. Remember the folder is not visible before you have followed the steps in Chapter 2.

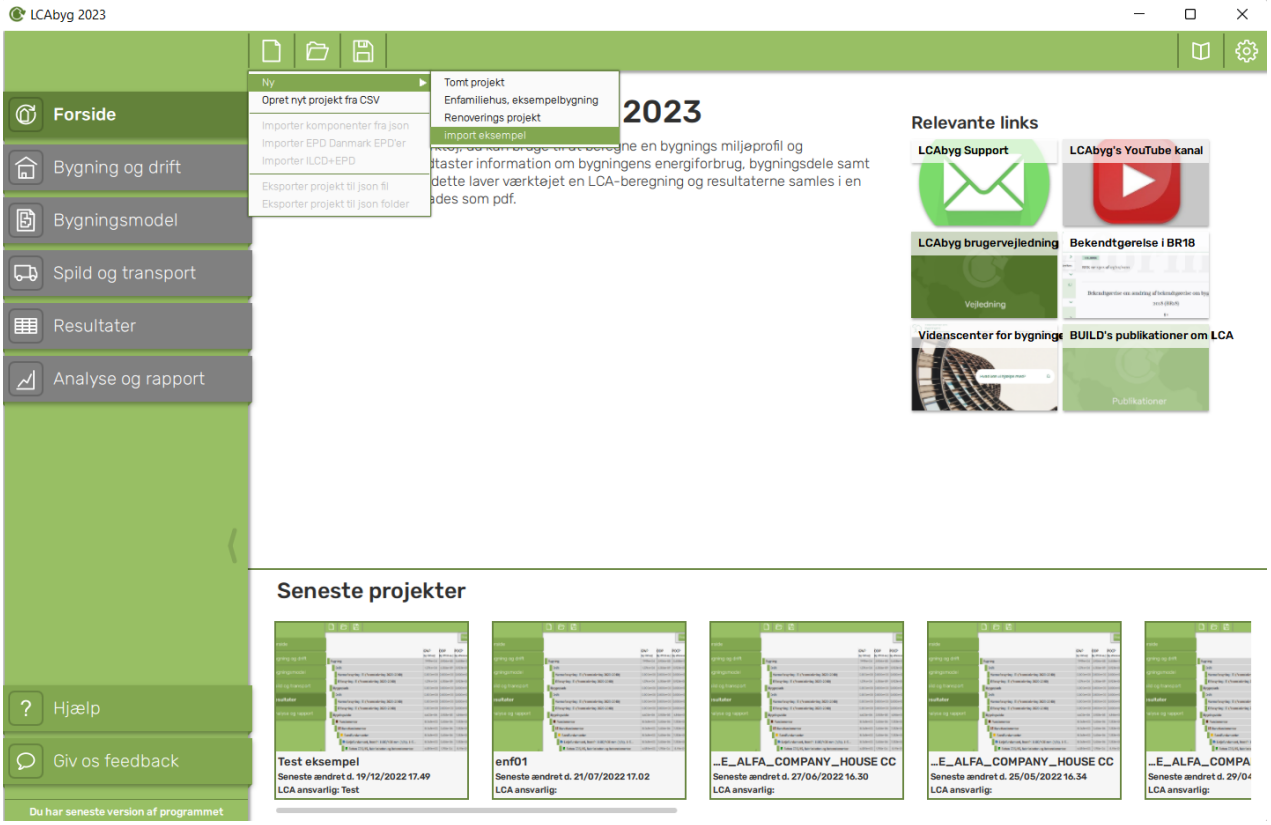


Figure 1.3: Project template the user can work in in LCAbyg

The project template, import_example, is located in the zip file when downloading the JSON guide. The import_example folder holds several json files, which you can work in and modify, see Figure 1.4.

Before you start working on the JSON files, download a text editor where you can work in the JSON format. We recommend the text editor Notepad++ which can be found at https://notepad-plus-plus.org/downloads/v7.9/ or Kate (if using a Mac computer) which can be found at https://kate-editor.org/.

Figure 1.4: Where to find the import_examples

1.2 Guide build up

The JSON guide for LCAbyg is divided into five main chapters each containing essential information and knowledge about the JSON files behind LCAbyg. Appendix A gives a detailed overview of the possible nodes and edges for an LCAbyg JSON project.

Chapter 1, *Introduction and Guide build up*, introduces the guide and the guide build up. This chapter also explains the basics of the JSON files and format.

Chapter 2, *Applying the basics - a step-by-step guide*, introduces the basics for the JSON files for LCAbyg, e.g. how Nodes and Edges work, the folder and files structure, as well as the Nodes and Edges, build up in LCAbyg.

Chapter 3, *The Building Operation and Energy Consumption*, focuses on how to create Nodes and Edges within the operation and energy consumption, e.g. utility sources (forsyningskilder) hereby electricity supply (elforsyning), heat supply (varmeforsyning), etc.

Chapter 4, *The Building Model*, focuses on how to create Nodes and Edges within the Building model, e.g. elements (bygningsdele), constructions (konstruktioner), products (bygningsvarer), stages (faser), etc.

Chapter 5, *Importing JSON*, explains how to work to import JSON projects with one or more scenarios, individual JSON components, and their respective results and computed quantities.

Appendix A.1, *Detailed overview*, shows a diagram of how entire LCAbyg is structured in JSON. Not all nodes and edges listed in this appendix are needed to model a project in the LCAbyg JSON format.

Appendix A.2, *JSON files*, shows a full list of all the JSON files, from which, the content can be copy-pasted. These codes are identical to the JSON files found in the import_example.

If you want to read about how to export JSON files and results, please read the guide ***JSON export guide)***, which can be found here: https://www.lcabyg.dk/da/usermanual/brugervejledning-andre-vaerktojer/ If you want to know more about the scenario function in the JSON format, please read the guide ***JSON scenarios guide***, which can be found here: https://www.lcabyg.dk/da/usermanual/brugervejledning-andre-vaerktojer/

1.3 Understanding the basics

The json files consist of lists that contain several objects which hold a string of Nodes and Edges, see Figure 1.5.

When creating a json file, it is important to keep track of the essential characters and indentation, as well as their location in the file. Section 1.3.1 and Section 1.3.2 will review which characters are essential when creating a list and a dictionary.

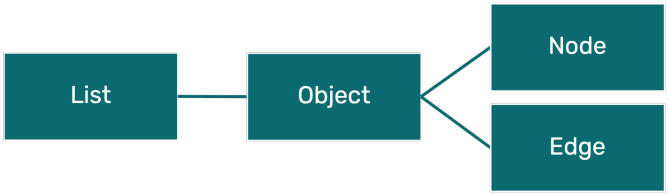


Figure 1.5: Simplification of the json files built up

1.3.1 Lists

The json files consist of lists. Start the list in the json file by adding "[" in the beginning and end the code with "]" in the end. If you want to add multiple Nodes and Edges in the same code/file this is separated by a comma ",". The last Node or Edge must never contain a comma. The order of Nodes and Edges does not matter.

Example

```
[
  Node1,
  Edge1,
  Node2
]
```

1.3.2 Dictionary

The json files contain several dictionaries. A dictionary is an unordered collection of data values, used to store data values like a map. The dictionaries are denoted by " " and hold a key/value pair. The keys must be strings and separated from their associated values with a colon.

Example

```
{
 "Key1": Value1,
 "Key2": Value2
}
```

1.3.3 Nodes and Edges

The json files consist of Nodes and Edges. A Node can be explained as a visual representation of an entity, whereas an edge is a visual representation of a relation. An Edge can connect two Nodes, as illustrated in Figure 1.6.

Figure 1.6: How an Edge can connect two Nodes

1.3.4 ID's

Each Node and Edge contains a unique ID. The ID's follow the UUID standard [1]. The ID must only be used once when creating new Nodes and Edges. These IDs are later used to refer to each other e.g. when referring to an Edge between two Nodes.

1.3.5 Find ID's and names from gen_dk

When creating a project template, it is possible to refer to the LCAbyg library (gen_dk), which can be found in the folder gen_dk located in downloaded import_example folder, see Figure 1.7. The folder contains all the IDs and names of the elements, constructions, building products, operation utilities, etc. which can be found in LCAbyg's library. The IDs are used when e.g. making an Edge between an existing Node and a new Node.

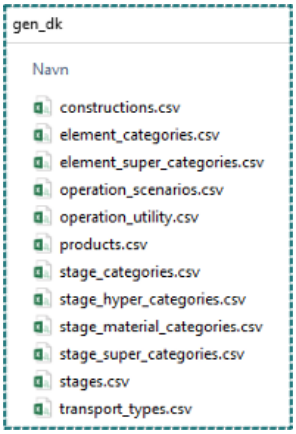


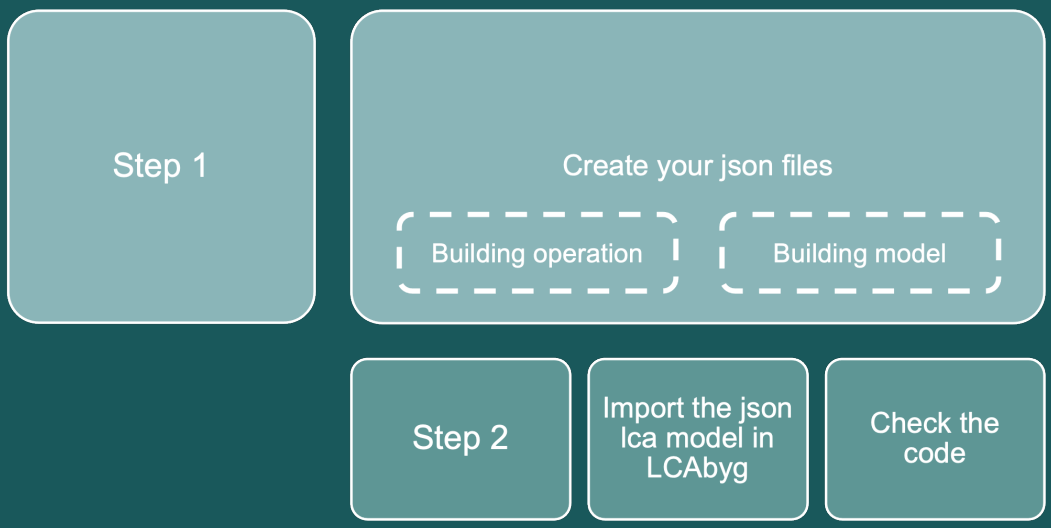
Figure 1.7: The csv files located in the gen_dk folder

[1] https://en.wikipedia.org/wiki/Universally_unique_identifier

Chapter 2

Applying the basics - a step by step guide

Apply the basics from what you learned in the earlier chapter. This chapter contains a step-by-step guide on how to set up your json files before making new Nodes and Edges within the Building Model, the Building Operation and Energy Consumption and how to import and export json files in LCAbyg.



2.1 Creating json files

There are multiple steps in the creation of json files. The following sections will review each step.

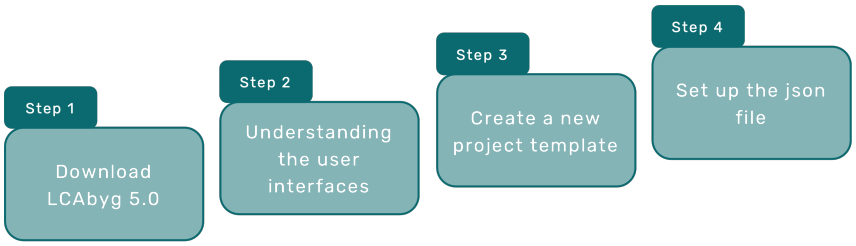


Figure 2.1: Step-by-step guide for creating json files

Step 1 - Download LCAbyg

Download the newest version of LCAbyg at https://www.lcabyg.dk/da/download-legacy/. You have to create a user before you can download the program. If you are using a Mac computer please read the Download Guide for Mac at https://www.lcabyg.dk/da.

If you have an older version of LCAbyg, please make sure to download the newest version and open the newest version of LCAbyg before working in the program.

Step 2 - Understand the user interface

Before creating constructions, building products, etc. in json files, a basic understanding of the user interface in LCAbyg must be established. This guide only touches on a fraction of the understanding of the user interface and a deeper understanding must be achieved when reading the LCAbyg guide version 5. Figure 2.2 illustrates the basics of the user interfaces in LCAbyg 5.

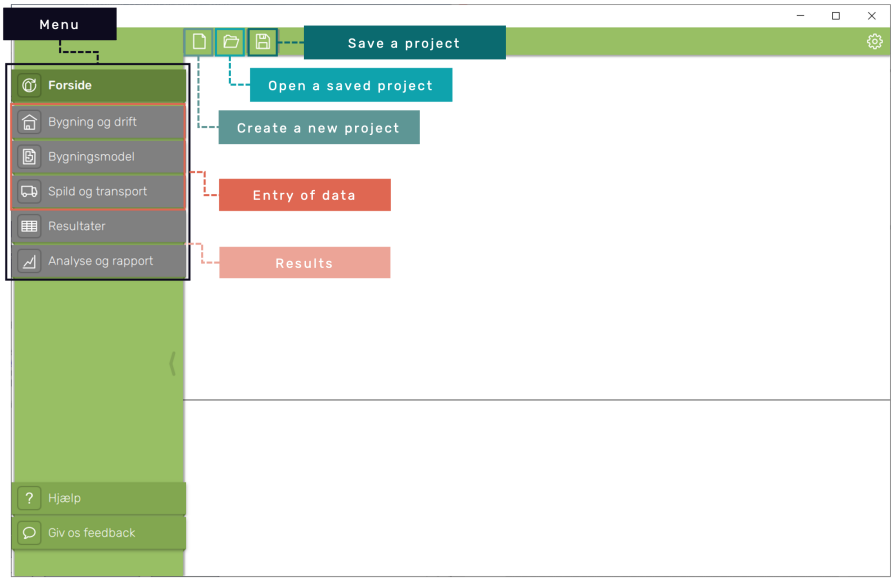


Figure 2.2: User interface in LCAbyg 5

Step 3 - Create a new project template

You can create a new project template like the empty project and case1. By creating a new project template, you can change e.g., the defaults for a project in LCAbyg, which can be beneficial if you intend to use the same template for a project over and over.

In this example, the project template named import_example has been created, which can be used and modified by the user. The project template already contains serval json files that can be modified and added. There are two ways to import the import_example in LCAbyg, which is described in Table 2.1.

Administration rights on the computer	No administrator rights on your computer
Follow step 3.1, 3.1.1, 3.2, 3.3, 3.4	Follow step 3.1, 3.1.2-3.13, 3.2, 3.3, 3.4

Table 2.1: The two ways to import import_example

Step 3.1 - Find the LCAbyg 5 folder

The first step in the creation of a new project template is to find the LCAbyg 5 folder. When downloading LCAbyg you chose the location of the folder. In the vast majority of cases, the folder is located in **C:\Program Files\SBi\LCAbyg 5 (64 bit) (5.3.1.0)***, see Figure 2.3. If you are using a Mac computer, please go into programs, left click on the LCAbyg logo, click "vis indholdet af pakken", click on the contents folder, and then click on the MacOS folder.

**The last number is the version number and change when you download newer versions of the program.*

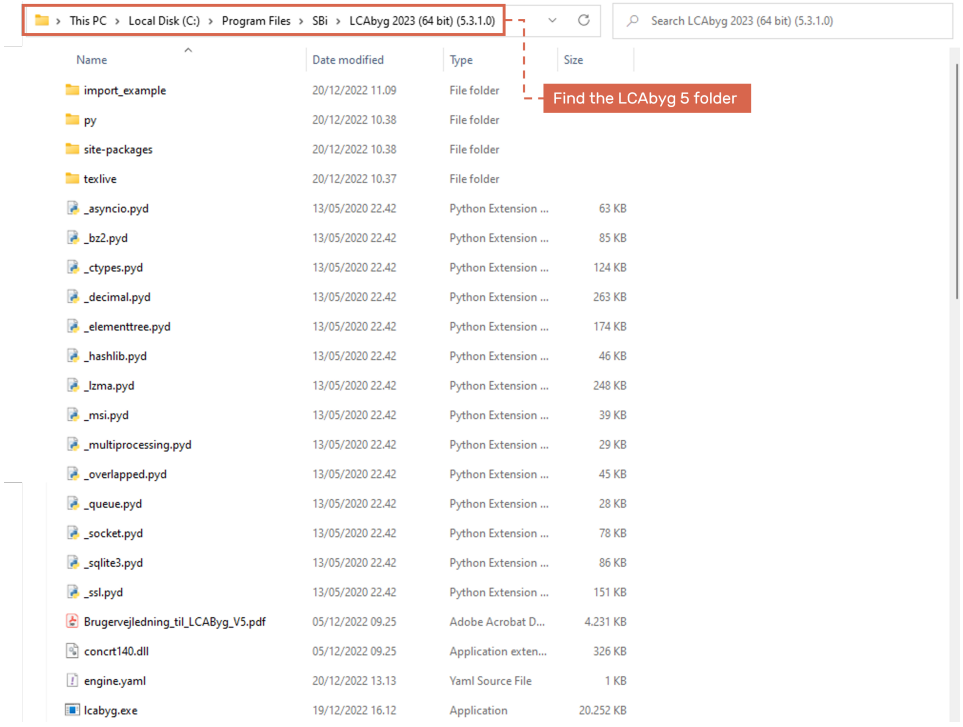
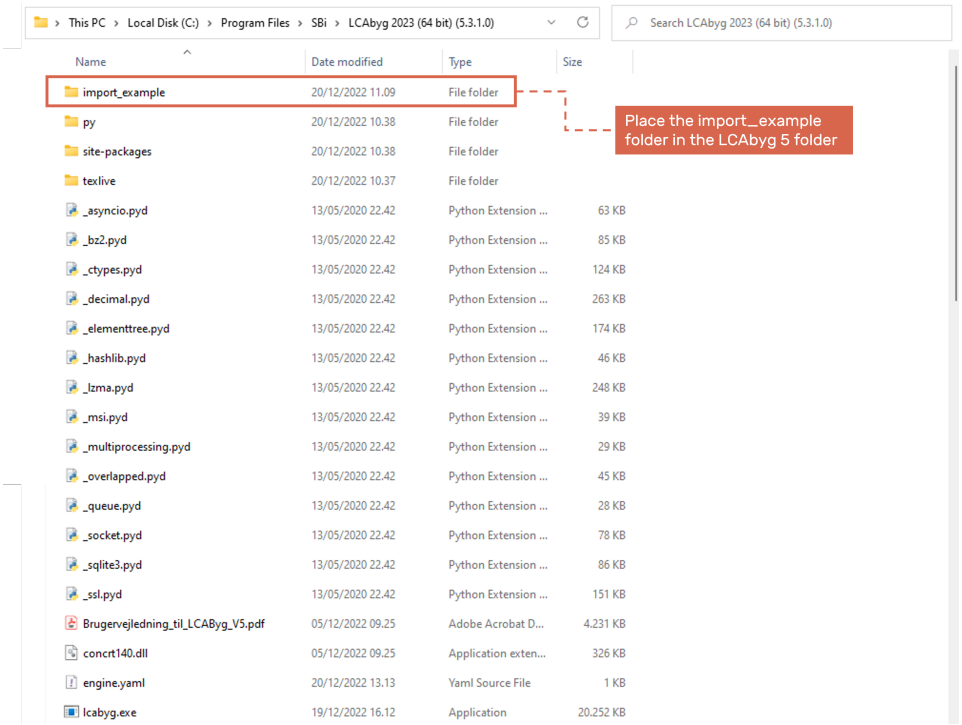


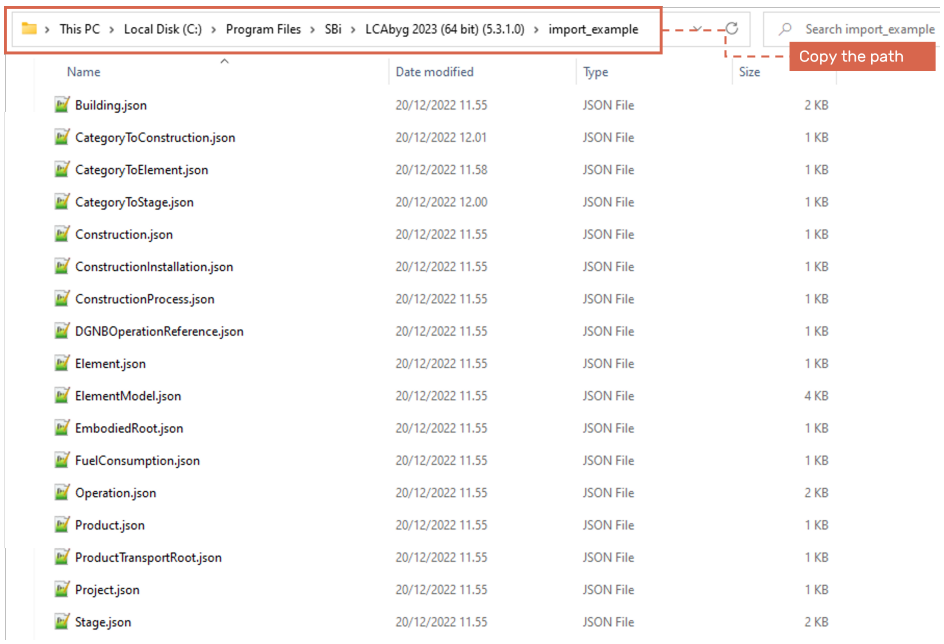
Figure 2.3: Find the LCAbyg 5 folder on a Windows computer

Step 3.1.1 - Administration rights on the computer

After locating the LCAbyg 5 folder, place the project template folder named import_example in the LCAbyg 5 folder to obtain the same path. The import_example folder is located in the zip file together with this JSON Guide (see Section 1.1.2). **Unzip** the import_example folder and **open** the **import_example folder** and **copy the folder path**. Inset the copied folder path in your Notepad or a Word document as you will need this path later in Step 3.2.

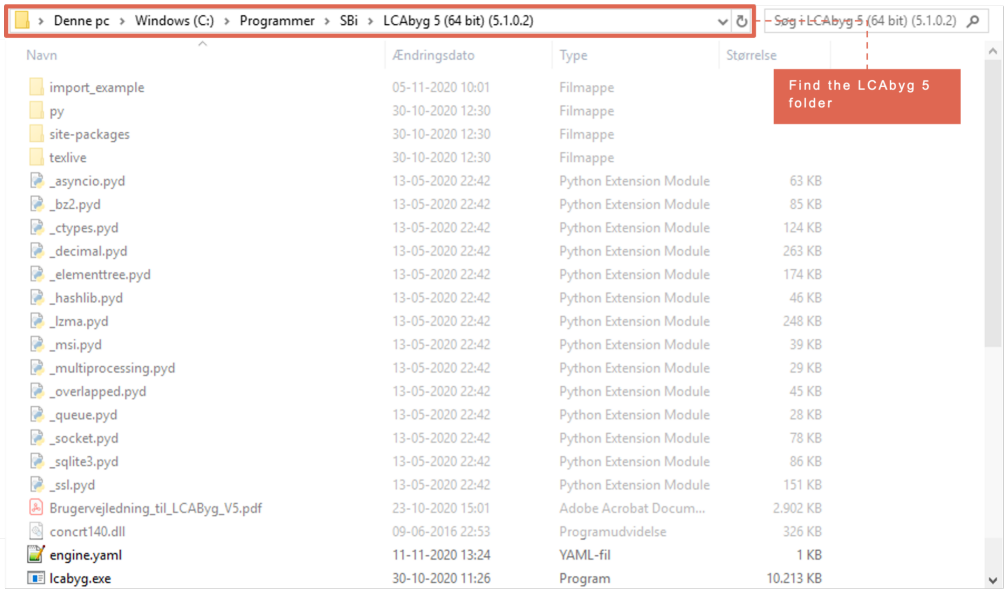
Note: You can create your project template by creating a folder like import_example, name the folder and locate it in the LCAbyg 5 folder. We recommend working with the import_example, to begin with.





Step 3.1.2 - No administrator rights on the computer

If you do not have administrator rights, please follow the following steps. Copy the engine.yaml file from LCAbyg 5 folder in which you found in Step 3.1.

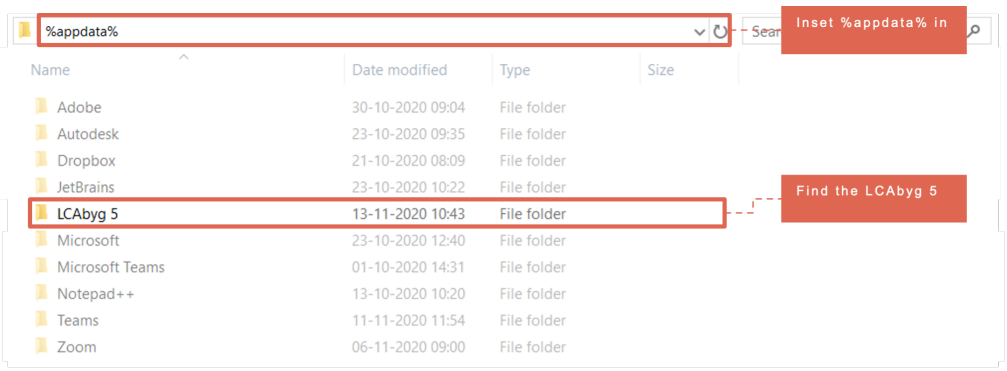


Step 3.1.3 - Find the LCAbyg 5 folder

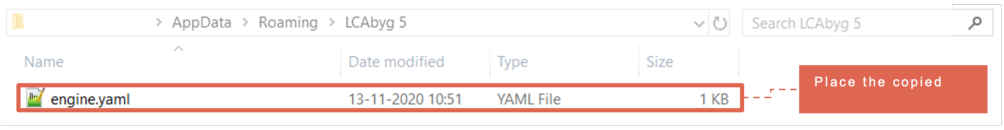
In your file explorer, write %appdata% in address bar. The folder named LCAbyg 5 (not the same folder as in Step 3.1) will appear.

Open the LCAbyg 5 folder and place the copied engine.yaml file in the folder. If you are using a Mac computer you must do the following steps, to find the %appdata% folder:

1. Open Finder.

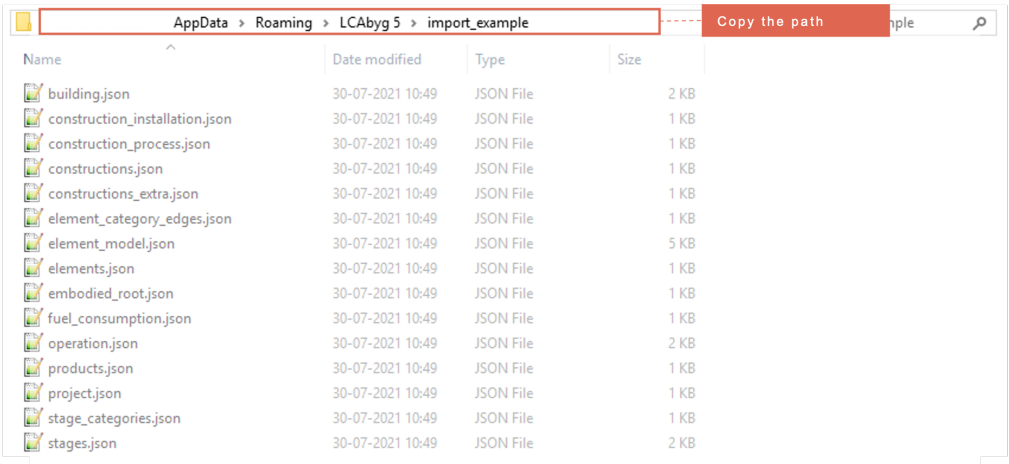


2. Click "Go" on the menu bar.
3. Press and hold the "option/alt" key.
4. Click the "Library" shortcut which appears.
5. Click on the folder Application Support
6. Create the folder "lcabyg5" and dump engine.yaml file



In the same folder as your placed the engine.yaml you must place import_example in the LCAbyg 5 folder.

Open the import_example folder and **copy the folder path**. Inset the copied folder path in your Notepad or a Word document as you will need this path later in Step 3.2.



Step 3.2 - Open engine.yaml file in a text editor

Open the file engine.yaml in a text editor. Remember if you have no administrator rights you have placed the engine.yaml file in another LCAbyg 5 folder, see Figure 2.5.

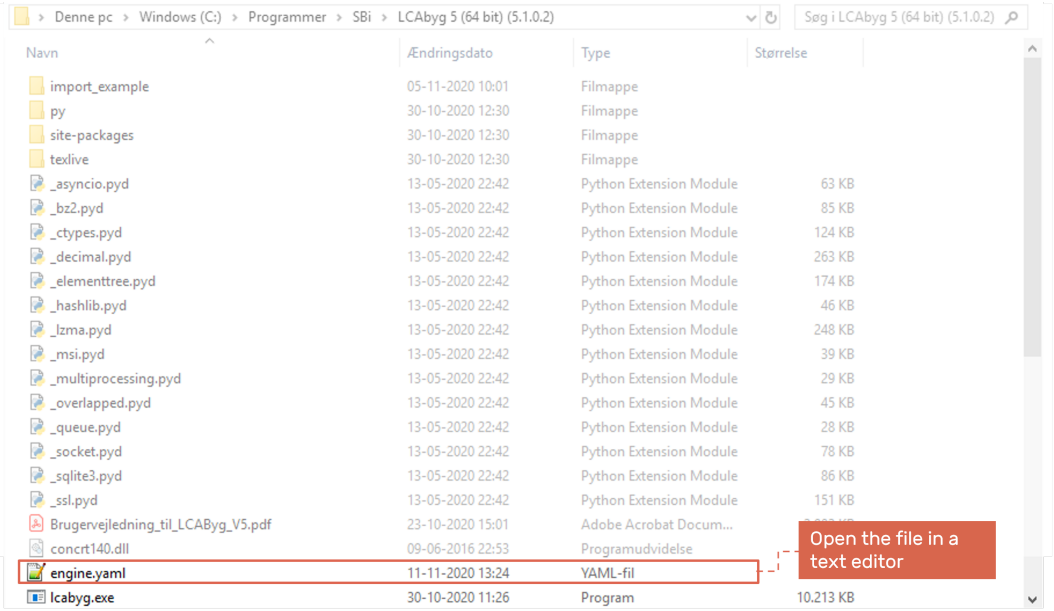


Figure 2.4: Find and open the engine.yaml (With administrator rights).

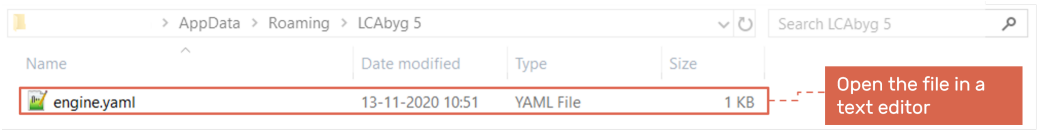


Figure 2.5: Find and open the engine.yaml (No administrator rights).

Step 3.3 - Copy and paste an existing template in engine.yaml file

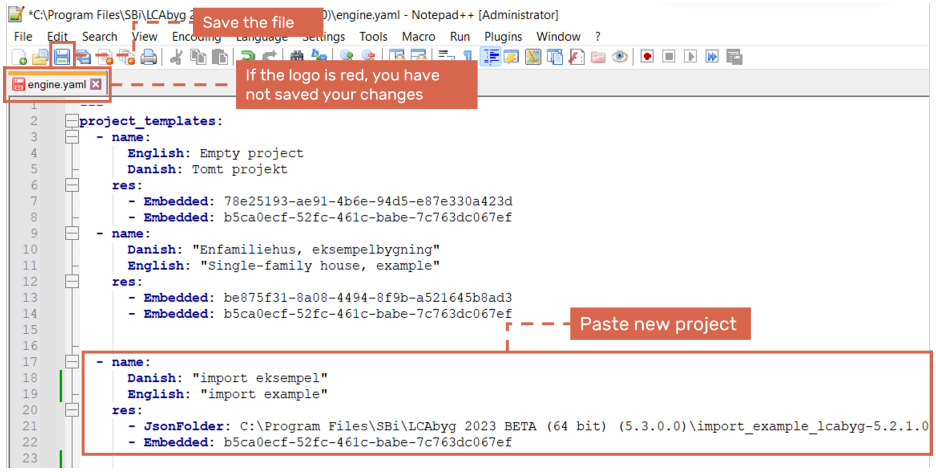
After opening the **engine.yaml** file you must create a new project template. Copy the following text in the box beneath and place it after the existing project templates in the engine.yaml file. Replace the ***Json-Folder*** path with the folder path you copied and placed in your Notepad or Word document from Step 3.1.3.

```
1  ---
2  project_templates:
3    - name:
4        English: Empty project
5        Danish: Tomt projekt
6      res:
7        - Embedded: 78e25193-ae91-4b6e-94d5-e87e330a423d
8        - Embedded: b5ca0ecf-52fc-461c-babe-7c763dc067ef
9    - name:
10       Danish: "Enfamiliehus, eksempelbygning"
11       English: "Single-family house, example"
12     res:
13       - Embedded: be875f31-8a08-4494-8f9b-a521645b8ad3
14       - Embedded: b5ca0ecf-52fc-461c-babe-7c763dc067ef
15
16   - name:
17       Danish: "Test"
18       English: "Test"
19     res:
20       - JsonFolder: C:\Program Files\SBi\LCAbyg 5 (64 bit) (5.3.0.0)\import_example
21       - Embedded: b5ca0ecf-52fc-461c-babe-7c763dc067ef
```

Listing 2.1: engine.yalm

If you want to change the name of the project template, enter a new name in the quotation marks (“ ”) in line 17 and 18. Remember the folder path created in Step 3.1.1 or 3.1.2 should be placed in line 20.

Remember to save the **engine.yaml** file after placing and editing the new project template. If the save logo turns red, you have not saved the changes in the file. Save the file by clicking on the save logo in the menu.



Step 3.4 - Check if the new project template appears in LCAbyg

Check if the new project template appears in LCAbyg by opening the program as usual. Click on the **“Opret nyt project”** logo in the menu in the upper left corner. Click on “Ny” and then check if your new project template appears.

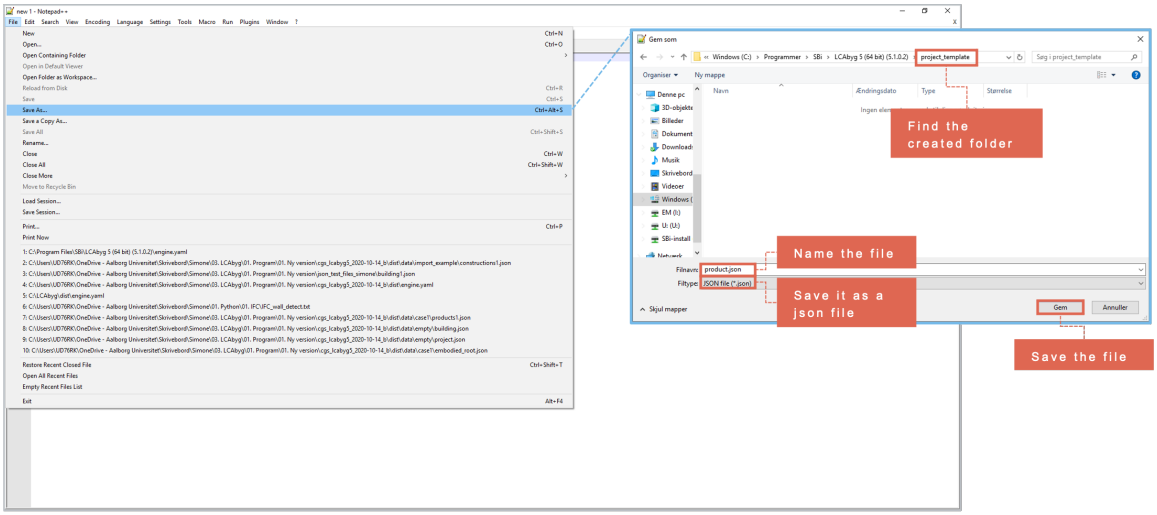
Note: If you have created your own project template and therefore not used the import_example an error will appear and tell that the program is missing a node.

Note: You can also open a new project using the debug version of LCAbyg named “lcabyg_debug.exe”. Read more about how to troubleshoot in Appendix D – Troubleshooting.

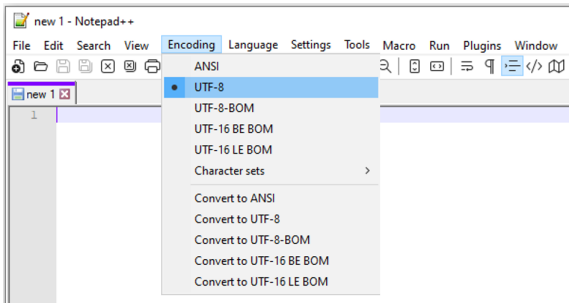
Step 4 - Set up the JSON file

Step 4.1 - Set up a JSON file

Start by **opening your text editor.** Click on the file and then **Save As**. **Name** your file with the extension ".json" and **save** your file as a **JSON file (*.json)** in the **folder** named **import_example** which is located in the LCAbyg 5 folder. If you have created a new project template folder, you must locate your JSON files in the new folder.

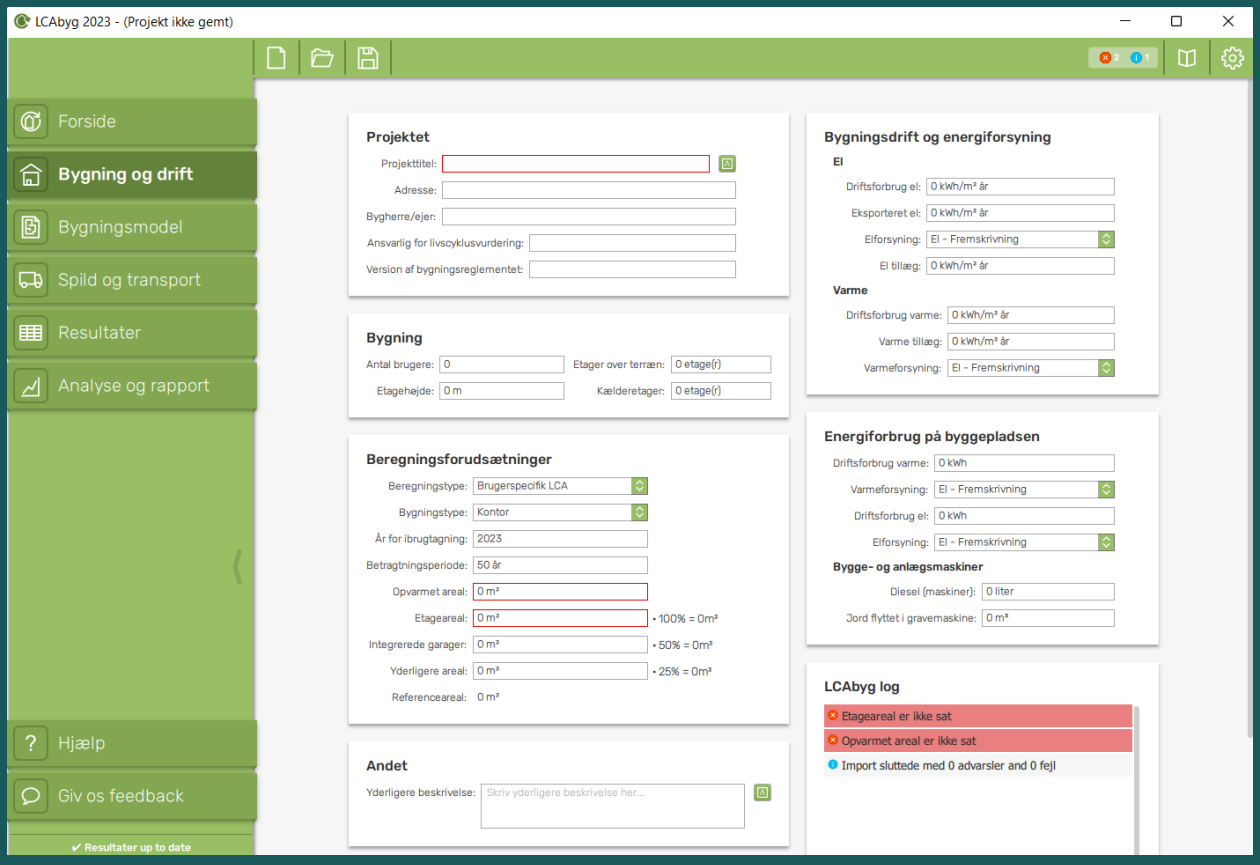


Note: Remember to set the Encoding default to UTF-8. You will find the encoding setting in the top bar 'Encoding'.



Chapter 3

The Building Operation and Energy Consumption



3.1 Minimum requirements

The minimum requirements for the building operation and energy consumption can be found in the simplified Figure 3.1. You can find the full overview of the all Nodes and Edges in Appendix A.

When creating an Edges between e.g. the Building and Operation you must also create Edges between the Operation and Operation Utility as well as the Operation Utility and the Operation Scenario.

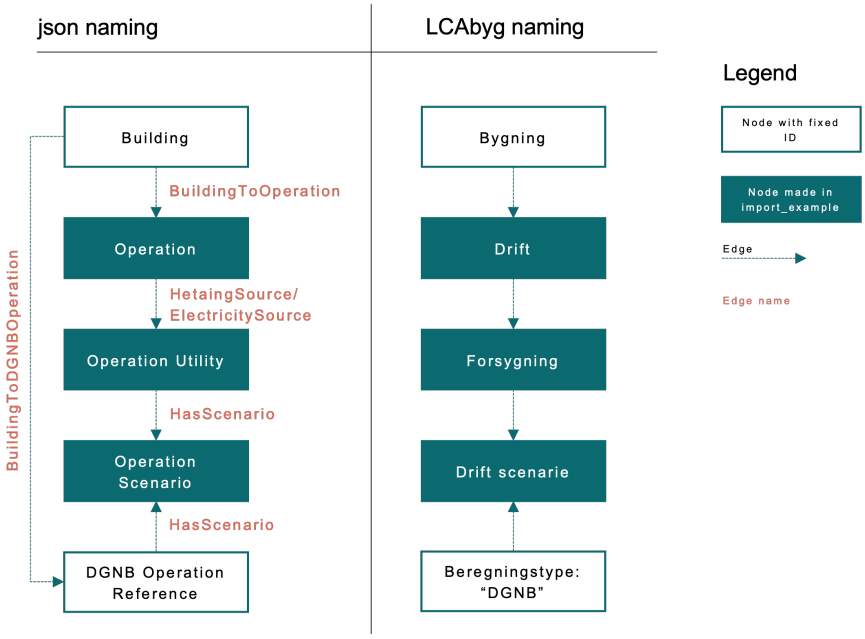


Figure 3.1: Edges required for the building and operation

If you want to connect new Nodes or connect a new Node with an existing Node from the gen_dk library a new Edge must be created, see Figure Figure 3.2.

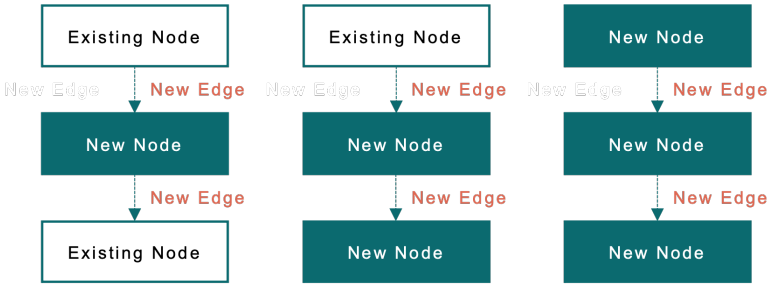


Figure 3.2: Examples on connections between Nodes and Edge

3.2 The creation of new nodes and edges

In the following sections, it is presented how to make and set up new nodes for operation, utilities etc. as well as how to create edges between them. The code can be copied directly from the JSON files in Appendix B. Remember to save all JSON files in the import_example folder, see Figure 2.4 and Figure 2.5 (depending on whether or not you have administrator rights)

Figure 3.3: Setup example for template for Nodes and Edges

A good rule of thumb

In most cases, you can look at edge names and see which order you need to refer to the ID numbers. E.g. in the example above where we have an edge named Building To Operation. The first id you refer to is Building ID number where the next ones are Operation ID number.

3.2.1 Creating a new operation node and edge to ElectricitySource and HeatingSource (Operation)

When creating a new operation, an edge between an operation and operation utility must be created for both the edge HeatingSource and the edge ElectricitySource.

Before creating the node, a list that contains a dictionary is created, as illustrated in Figure 1.5. After this step, a new node can be created in the JSON file.

In the following table, an example of how to create a new operation node is explained. You can use the example as a template when creating a new operation node in your JSON file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.1 or in the import_example (Operation.json).

Example	Comment
[	Begin list
{	Begin dictionary
"Node": {	Begin node
"Operation": {	Insert unique ID
"id": "Inset_unique_operation_ID_number",	Create and inset unique ID
"electricity_usage": 17.3, "heat_usage": 0, "electricity_production": 0	Inset values for the usages and production.
}	End new operation
}	End node
},	End dictionary and continue list
{	Begin dictionary
"Edge": [	Begin edge
{	
"ElectricitySource": " Inset_unique_ID_number "	Inset a new ID
},	End new edge
"Inset_Operation_ID_number ",	Inset the operation ID
"Inset_OperationUtility _ID_number"	Inset operation utility ID from gen_dk library or a unique ID
]	End edge
},	End dictionary and continue list
{	Begin dictionary
"Edge": [	Begin edge
{	
"HeatingSource": " Inset_unique_ID_number "	Inset a new ID
},	End new edge
"Inset_Operation_ID_number ",	Inset the operation ID
"Inset_ OperationUtility _ID_number"	Inset operation utility ID from gen_dk library or a unique ID
]	End edge
}	End dictionary
]	End list

3.2.2 Creating a new edge between building and operation node (BuildingToOperation)

An edge between a building and an operation must be created.

Before creating the edge, a list that contains a dictionary is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the JSON file.

In the following table, an example of how to create a new edge between the building and operation is explained. You can use the example as a template when creating a new edge in your JSON file. Please note that the text highlighted with yellow in the example requires your input. Find the full JSON file in Appendix B.1 or in the import_example (Building.json).

Example	Comment
[	Begin list
{	Begin dictionary
"Edge": [	Begin edge
{	
"BuildingToOperation": "Inset_unique_edge_ID_number"	Begin new edge* Create and inset unique ID
},	End new edge
"Inset_Building_ID_number ",	Inset the building ID from the building.json file which can be found in the import_example folder
"Inset_Operation_ID_number"	Inset either existing operation ID or new operation ID
]	End edge
}	End dictionary
]	End list

3.2.3 Creating a new operation utility node (OperationUtility)

Start by creating a new JSON file and save it in the import_example.

Utility type can either be set to Electricity or Heating for calculation method "Normal". The other calculation methods "DNGB" and "SC" (Sustainability Class) specific utility types are required and are thus hardcoded in the LCAbyg. See Appendix C.2 for detailed explanation of Calculation Method and Operation Utility.

Before creating the node, a list that contains a dictionary is created, as illustrated in Figure 1.5. After this step, a new node can be created in the JSON file.

In the following table, an example of how to create a new operation utility node is explained. You can use the example as a template when creating a new operation utility node in your JSON file. Please note that the text highlighted with yellow in the example requires your input. Find the full JSON files in Appendix B.1 or in the import_example (OperationUtility).

Example	Comment
[	Begin list
{	Begin dictionary
"Node": {	Begin node
"OperationUtility": {	Begin new operation utility
"id": "Inset_unique_utility_ID_number",	Create and inset unique ID
"name": {	Create and inset a unique name. Remember the Danish, English, German must differently from each other.
"Danish": "Inset_ unique_utility_name"	
"English": "Inset_ unique_utility_name1"	
"German": "Inset_ unique_utility_name2"	
},	
"util_type": "Insert_type_of utility ",	Inset either "Electricity" "Heating" or "Other"
"data_points": {	Inset data for utility
"2020": {	Inset year and inset data for each indicator
"GWP": 0.264,	
"ODP": 4.89e-14,	
"POCP": 7.02e-05,	
"AP": 0.00066,	
"EP": 0.000141,	
"ADPE": 9.24e-08,	
"ADPF": 2.75,	
"PENR": 2.85,	
"PER": 6.92,	
"SENR": 3.61e-08,	
"SER": 0.000658	
}	
}	
}	End new operation utility
}	End node
}	End dictionary
]	End list

3.2.4 Creating a new operation scenario node (OperationScenario)

Start by creating a new json file and save it in the import_example.

Before creating the node, a list that contains a dictionary is created, as illustrated in Figure 1.5. After this step, a new node can be created in the json file.

In the following table, an example of how to create a new operation scenario node is explained. You can use the example as a template when creating a new operation scenario node in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.1 or in the import_example (OperationScenario).

Example	Comment
[	Begin list
{	Begin dictionary
"Node": {	Begin node
"OperationScenario": {	Begin new operation scenario
"id": "Insert_Operation_ID_number",	Create and inset unique ID DGNBOperation Reference)
"name": {	Create and inset a unique name. Remember the Danish, English, and German names must be different from each other
"Danish": "Insert_ unique_scenario_name"	
"English": "Insert_ unique_utility_name1"	
"German": "Insert_ unique_utility_name2"	
},	
"description": "Description for scenario",	Inset description for scenario
}	End new operation scenario
}	End node
}	End dictionary
]	Begin list

3.2.5 Creating a new edges between operation utility and operation scenarios node (HasScenario)

An edge between an operation utility and operation scenarios must be created.

Start by creating a new json file and save it in the import_example folder.

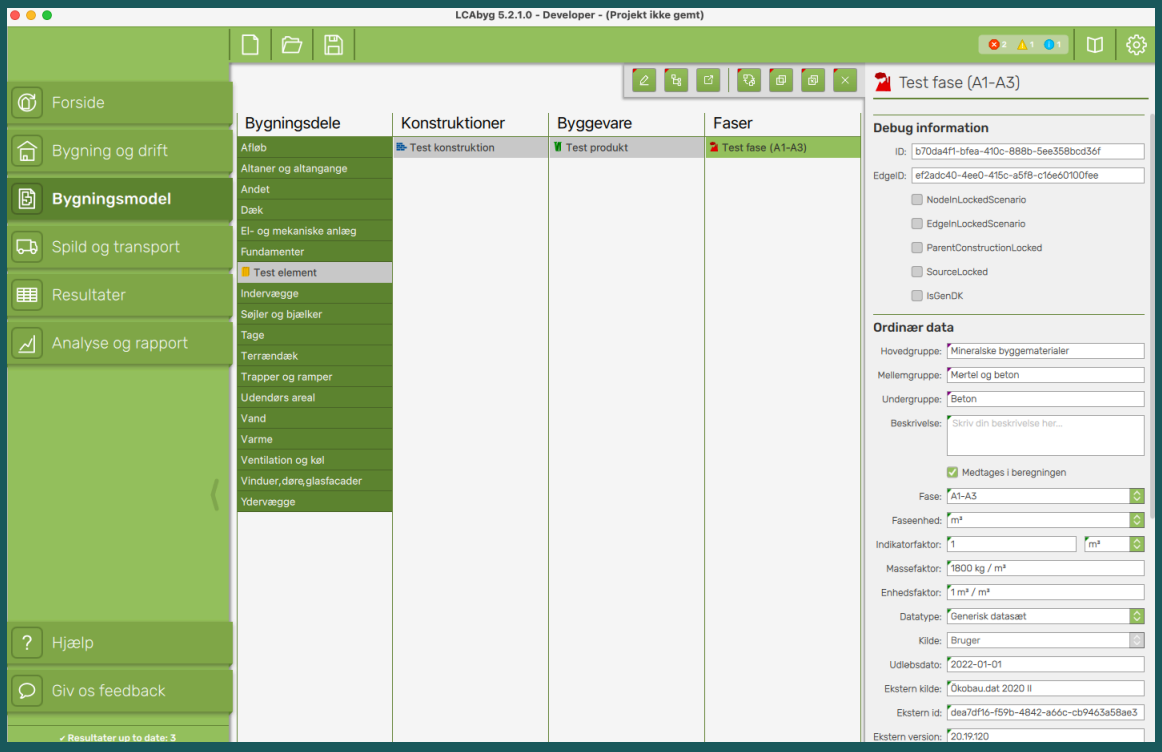
Before creating the edge, a list that contains a dictionary is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between the building and operation is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.1 or in the import_example (HasScenario).

Example	Comment
[	Begin list
{	Begin dictionary
"Edge": [	Begin edge
{	
"HasScenario": "Inset_unique_edge_ID_number"	Begin new edge Create and inset unique ID
},	End new edge
" Insert_OperationUtility_ID_number ",	Insert operation utility id
" Insert_OperationScenario_ID_number "	Inset operation scenario ID
]	End edge
}	End dictionary
]	End list

Chapter 4

The Building Model



4.1 Minimum requirements

The minimum requirement can be found in Figure 4.1. You can find the full overview of the all Nodes and Edges in Appendix A.

When creating an edge between e.g. the Project and the Building you must create an edge between e.g. a Stage Category and a new Stage or an Element Category and a new Element. You must use the Stage Categories and Element Categories ID's, which can be found in the library of LCAbyg in the gen_dk folder located in the zip files with the json guide.

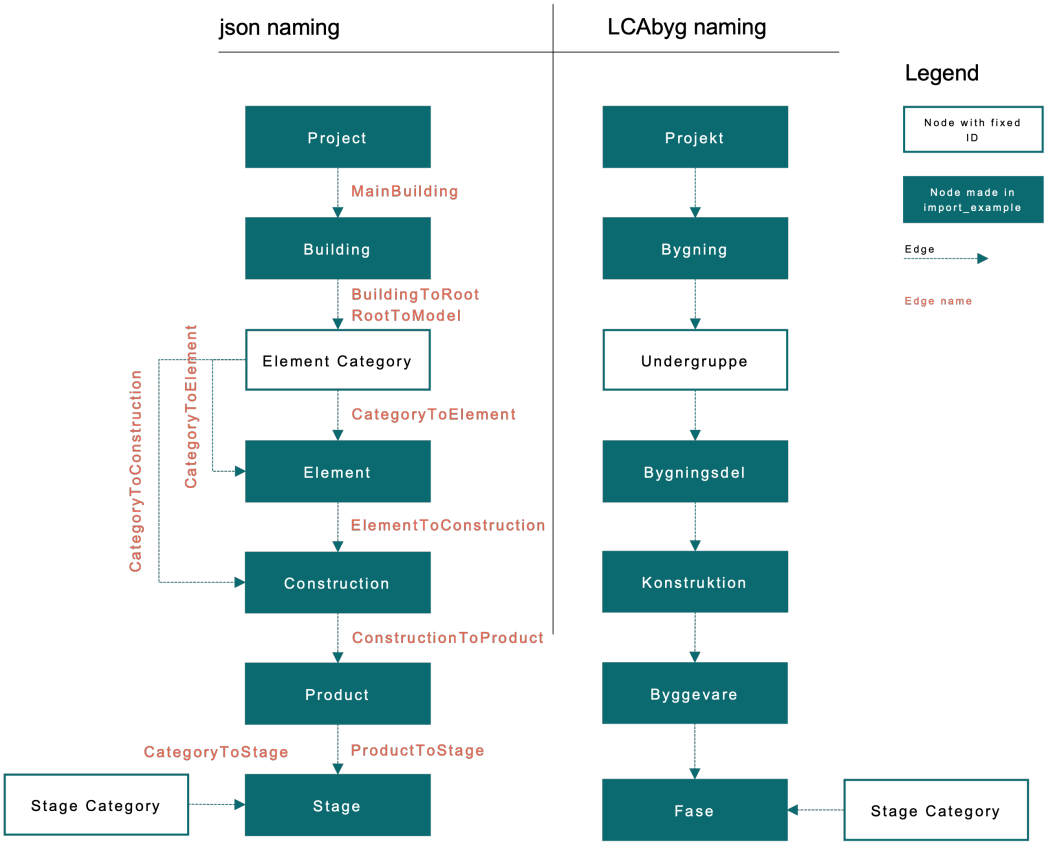


Figure 4.1: Edges required for the building model

4.1.1 Find ID's and names from gen_dk

The LCAbyg library (gen_dk) can be found in the folder gen_dk located in the LCAbyg 5 folder. The folder contains the IDs and names of all the elements, constructions, building products, operation utilities, etc. which can be found in the library of LCAbyg. The IDs are used when e.g. making an Edge between an existing Node and a new Node.

4.2 The creation of new nodes and edges

In the following sections, it is presented how to make and set up new nodes for elements, constructions etc. as well as how to create edges between them.

Example	Comment
[	
{	Begin dictionary
"Node": {	Begin node
"Construction": {	Begin new construction
"id": "Inset_unique_construction_ID_number",	Create and inset unique ID
"name": {	Create and inset a unique name
"Danish": "Inset_ unique_construction_name"	
},	
"unit": "M2",	Set the unit (kg, m, m2, m3 and stk)
"source": "User",	Inset the source
"comment": "Write_a_comment_if_needed",	Inset a comment*
"layer": 1,	Inset the layer (1,2 or 3)**
"locked": true	
}	End new construction
}	End node
}	End dictionary
]	

Figure 4.2: Setup example for template for Nodes and Edges

4.2.1 Creating a new project node (Project)

Start by creating a new json file and save it in the import_example.

Before creating the node, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new node can be created in the json file.

In the following table, an example of how to create a new project node is explained. You can use the example as a template when creating a new project in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (Project.json).

Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"Project": {	Begin new element
"id": "Insert_unique_project_ID_number",	Create and inset unique ID
"name": {	
"Danish": "Test eksempel"	
},	
"address": "Testvej 1, 1111 Testby",	
"owner": "Test",	
"lca_advisor": "Test",	
"building_regulation_version": "BR2018"	
}	End new element
}	End node
}	End object
]	End list

4.2.2 Creating a new building node (Building)

Start by creating a new json file and save it in the import_example.

Before creating the node, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new node can be created in the json file.

In the following table, an example of how to create a new building node is explained. You can use the example as a template when creating a new element in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (Building.json).

Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"Building": {	Begin new node*
"id": "Insert_unique_edge_ID_number",	Create and insert unique ID
"scenario_name": "Original bygningsmodel",	Default value if no scenarios has been activated.
"locked": "Unlocked",	Setting for whether the building scenario can be edited or not
"description": "Test",	Insert description (optional)
"building_type": "Other",	Insert building type
"heated_floor_area": 0,	Insert heated floor area (square meters)
"gross_area": 0,	Insert gross area (square meters)
"gross_area_above_ground": 0,	Insert gross area above ground (square meters)
"storeys_above_ground": 0,	Insert stories above ground
"storeys_below_ground": 0,	Insert stories below ground
"storey_height": 0,	Insert storey height
"initial_year": 2015,	Insert initial year
"calculation_timespan": 50,	Insert calculation time span
"calculation_mode": "SC",	Insert calculation mode*
"outside_area": 0,	Insert plot area
"plot_area": 0,	Inset outside area
"energy_class": "LowEnergy"	Insert energy class
}	End new edge
}	End object
]	End list

*See Appendix C.2 for a detailed description of calculation modes.

4.2.3 Creating a new edge between project and building node (MainBuilding)

An edge between a new project and a building must be created.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a project and a building is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (MainBuilding.json).

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
"MainBuilding": "Insert_unique_building_ID_number",	Create and insert unique main building ID (different from building ID)
"Insert_unique_project_ID_number",	Insert project ID
"Insert_unique_building_ID_number"	Insert building ID (different from main building ID)
]	End edge
}	End object
]	End list

*Edge names can be found in Figure 4.1 or Appendix A

4.2.4 Creating a new edge between a new building and root node (BuildingToRoot)

An edge between a new building and a root must be created.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a building and an building root is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (BuildingToRoot.json).

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
“BuildingToRoot": "Insert_unique_edge_ID_number"	Create and inset unique ID
}	End new edge
},	
"Insert_building_ID_number",	Insert existing building id
"Insert_root_ID_number"	InseRt either existing root ID or new root ID
]	End edge
}	End object
]	End list

*Edge names can be found in Figure 4.1 or Appendix A

4.2.5 Creating a new embodied root node (EmbodiedRoot)

Start by creating a new json file and save it in the import_example.

In the following table, an example of how to create a new embodied root node is explained. You can use the example as a template when creating a new element in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (EmbodiedRoot.json).

Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"EmbodiedRoot": {	Begin new node*
"id": "Insert_unique_edge_ID_number",	Create and inset unique ID
}	End new edge
}	End object
]	End list

4.2.6 Creating a new edge between a embodied root and model (RootToModel)

An edge between the embodied root and the model must be created.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a embodied root and the model is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (EmbodiedRoot.json).

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
"RootToModel": "Insert_unique_edge_ID_number"	Create and insert unique ID
},	
"Insert_root_ID_number",	Insert root ID
"Insert_element_model_ID_number"	Insert element model ID*
]	End edge
}	End object
]	End list

4.2.7 Creating a new edge between a embodied root and construction process (RootToConstructionProcess)

An edge between the embodied root and the model must be created.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between an embodied root and the construction process is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (EmbodiedRoot.json).

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
“RootToConstructionProcess": "Insert_unique_edge_ID_number"	Create and inset unique ID
},	
"Insert_building_ID_number",	Insert building ID
"Insert_operation_ID_number"	Inset either operation ID
]	End edge
}	End object
]	End list

4.2.8 Creating a new edge between a new building and operation node (BuildingToOperation)

An edge between a new building and operation must be created as described in 3.2.2.

4.2.9 Create a new element node (Element)

Start by creating a new json file and save it in the import_example.

Before creating the node, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new element node is explained. You can use the example as a template when creating a new element in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (Element.json).

Note: An Edge needs to be connected between the Element Node and Element Category Node.

Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"Element": {	Begin new element
"id": "Insert_unique_element_ID_number",	Create and insert unique ID
"name": {	Create and inset a unique name. Remember the Danish, English and German must different form each other.
"Danish": "Insert_ unique_element_name"	
"English": "Insert_ unique_element_name1"	
"German": "Insert_ unique_element_name2"	
},	
"source": "User",	The source
"comment": "Write_a_comment_if_needed",	Insert a comment
"locked": true,	
"excluded_scenarios": []	The empty list indicates that the elements is included in the scenario.
}	End new element
}	End node
}	End object
]	End list

*Edge names can be found in Figure 4.1 or Appendix A

4.2.10 Create a new edge between ElementCategory and Element node (CategoryToElement

An edge between a new element and an element category must be created.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between an element and an element category is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (CategoryToElement.json).

Note: IDs and names for the category elements can be found in folder gen_dk in the csv file named element_categories.

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
"CategoryToElement": {	Begin new edge*
"id": "insert_unique_edge_ID_number",	Create and inset unique ID
"enabled": true,	
"excluded_scenarios": []	
}	End new edge
},	
"insert_element_category_ID_number",	Insert existing element category ID in excel file
"insert_element_ID_number"	Insert either existing element ID or new element ID
]	End edge
}	End object
]	End list

* Edge names can be found in Figure 4.1 or Appendix A

4.2.11 Create a new construction node (Construction)

Start by creating a new json file and save it in the import_example.

Before creating the node, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new node can be created in the json file.

In the following table, an example of how to create a new construction node is explained. You can use the example as a template when creating a new construction in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (Construction.json).

Note: No Edge needs to be connected between the construction node and the element or/and element category.

Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"Construction": {	Begin new construction
"id": "Insert_unique_construction_ID_number",	Create and inset unique ID
"name": { "Danish": "Insert_ unique_construction_name" },	Create and inset a unique name
"unit": "M2",	Set the unit using all caps (KG, M, M2, M3, and STK).
"source": "User",	The source
"comment": "Write_a_comment_if_needed",	Insert a comment, e.g., specifications regarding construction built up
"locked": true	This key has no function anymore and is in the process of being phased out
}	End new construction
}	End node
}	End object
]	End list

4.2.12 Create a new edge between element and construction node (ElementToConstruction

An edge between a construction and an element can be created but is not a requirement. Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a construction and an element is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (ElementToConstruction.json).

Note: No Edge needs to be connected between the construction node and the element or/and element category

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
"ElementToConstruction": {	Begin new edge*
"id": "Insert_unique_edge_ID_number",	Create and insert unique ID
"amount": 59.9,	Insert the amount
"enabled": true,	
"excluded_scenarios": []	
}	End new edge
},	
"Insert_element_ID_number",	Inset either existing element ID or new element ID
"Insert_construction_ID_number"	Insert either existing construction ID or new construction ID
]	End edge
}	End object
]	End list

* Edge names can be found in Figure 4.1 or Appendix A

4.2.13 Create a new edge between ElementCategory and construction node (CategoryToConstruction)

An edge between a new construction and an element category must be created.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a construction and an element category is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (CategoryToConstruction.json).

Note: IDs and names for the element category can be found in folder gen_dk in the csv file named element_categories (see Figure 1.7).
Note: No Edge needs to be connected between the construction node and the element or/and element category

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
"CategoryConstruction": {	Begin new edge*
"id": "insert_unique_edge_ID_number",	Create and insert unique ID
"layers": [	
1	
]	
}	End new edge
},	
"insert_element_category_ID_number",	Insert existing element category ID in excel file
"insert_construction_ID_number"	Insert either existing element ID or new element ID
]	End edge
}	End object
]	End list

* Edge names can be found in Figure 4.1 or Appendix A

4.2.14 Creating a new product node (Product)

Start to create a new json file and save it in the import_example.

Before creating the node, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new node can be created in the json file.

In the following table, an example of how to create a new product node is explained. You can use the example as a template when creating a new product in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (Product.json).

You can find detailed information about the uncertainty factors in the LCAbyg 5.2 user guide and in the guide "DGNB Certificerings system". Both guides can be downloaded from this page: https://www.lcabyg.dk/da/vejledning/brugervejledning-lcabyg-5/

Note: No Edge need to be connected between the product node and the construction node

Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"Product": {	Begin new product
"id": "Inset_unique_product_ID_number",	Create and inset unique ID
"name": { "Danish": "Inset_ unique_product_name" "English": "Inset_ unique_product_name1" "Danish": "Inset_ unique_product_name2" },	Create and inset a unique name. Remember the Danish, English and German must differently from each other.
"source": "User",	The source
"comment": "Write_a_comment_if_needed",	Inset a comment
"uncertainty_factor": 1.0,	See table in user guide "DGNB Certificerings system" found at the bottom of this page: *https://www.lcabyg.dk/da/vejledning/brugervejledning-lcabyg-5/*
"uncertainty_factor_dgnb": 1.3	The uncertainty factor for the data for DNGB using "Generic data" must be 1,3
}	End new product
}	End node
}	End object
]	End list

4.2.15 Create a new edge between construction and product node (ConstructionTo-Product

An edge between a product and construction can be created but is not a requirement.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a product and construction is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (ConstructionToProduct.json).

Note: IDs and names for the construction can be found in folder gen_dk in the excel file named constructions (see Figure 1.7).

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
"ConstructionToProduct": {	Begin new edge*
"id": "insert_unique_edge_ID_number",	Create and insert unique ID
"amount": 0.05,	Set the product quantity used per construction unit*
"unit": "KG",	Set the unit
"lifespan": 100,	Inset the lifespan for the product
"demolition": false,	Set if the product is demolished (false/true)**
"delayed_start": 0	Set the delayed start***
"enabled": true,	
"excluded_scenarios": []	
}	End new edge
},	
"insert_construction_ID_number",	Insert either existing construction ID or new construction ID
"insert_product_ID_number"	Insert either existing product ID or new product ID
]	End edge
}	End object
]	Begin list

* Edge names can be found in Figure 4.1 or Appendix A

** The demolition takes place either here and now or after a number of years if a delayed start is chosen

*** A delayed start is chosen when the product is demolished and added in the future

4.2.16 Create a new stage node (Stage)

Start by creating a new json file and save it in the import_example.

Before creating the node, a list which contains an object is created, as illustrated in Figure 1.5. After this step a now node can be created in the json file.

In the following table, an example of how to create a new stage node is explained. You can use the example as a template when creating a new product in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (Stage.json).

Example	Comment
[	
{	Begin dictionary
"Node": {	Begin node
"Stage": {	Begin new stage
"id": "Inset_unique_ID_number",	Create and inset unique ID
"name": { "Danish": "Inset_ unique_stage_name" "English": "Inset_ unique_stage_name1" "Danish": "Inset_ unique_stage_name2" },	Create and inset a unique name. Remember the Danish, English and German must be different form each other.*
"comment": "Write_a_comment_if_needed",	Inset a comment
"source": "User",	The source
"locked": true	
"valid_to": "2020-01-01",	Inset valid date
"stage": "A1to3",	Inset stage(s)
"stage_unit": "M2",	Set the unit**
"indicator_unit": "M2",	Inset unit from EDP
"stage_factor": 1.0,	Inset the ratio between the indicator unit and stage unit
"mass_factor": 50.05,	Inset mass per indicator unit
"indicator_factor": 1.0,	Inset stage factor from EDP***
"external_source": "Ökobau.dat 2020 II",	Inset external source
"external_id": "Inset_external_ID",	Inset external source ID****
"external_version": "00.00.011",	Inset external version
"external_url": "http://www.external_source.com"	Inset external url
"data_type": "Generic",	GenDK data types: Generic, Template, Representative, Average, Specific or ProjectSpecific For EPDdk data: Generic, Branche Specific or Product Specific
"indicators": { "GWP": 71.6, "ODP": 5.33e-07, "POCP": 0.0298, "AP": 0.455, "EP": 0.0576, "ADPE": 0.000197, "ADPF": 1000.0, "PENR": 2160.0, "PER": 67.9, "SENR": 0.0, "SER": 0.0	Inset indicators from EDP
}	End new stage
}	End node
}	End dictionary
]	

* Add the stage(s) in the unique name to make it easier for the user to navigate in the program, e.g.: Concrete (A1-A3), Concrete (C3), etc.

** Select one of the following units (written in all caps) to declare you product or material as KG, M, M2, M3, or STK (in English = pieces).

*** The quantity stated together with the declared unit.

**** See Appendix E.1.

4.2.17 Create a new edge between product and stage node (ProductToStage)

An edge between a product and stage must be created.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains an object is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a product and stage is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or in the import_example (ProductToStage.json).

Note: IDs and names for the stage can be found in folder gen_dk in the excel file named stages.

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
"ProductToStage": {	Begin new edge*
"id": "insert_unique_edge_ID_number",	Create and insert unique ID
"excluded_scenarios": [],	The empty list indicates that the elements is included in the scenario.
"enabled": true	
}	End new edge
},	
"inset_product_ID_number",	Insert either existing product ID or new product ID
"inset_stage_ID_number"	Inset existing stage ID or new stage ID
]	End edge
}	End object
]	End list

Edge names can be found in Figure 4.1 or Appendix A

4.2.18 Create a new edge between stage category and stage node (CategoryToStage)

An edge between a stage category and a stage must be created if you are modelling stages.

Start by creating a new json file and save it in the import_example folder. Before creating the edge, a list that contains a dictionary is created, as illustrated in Figure 1.5. After this step, a new edge can be created in the json file.

In the following table, an example of how to create a new edge between a stage category and stage is explained. You can use the example as a template when creating a new edge in your json file. Please note that the text highlighted with yellow in the example requires your input. Find the full json files in Appendix B.2 or import_example (CategoryToStage.json).

Note: IDs and names for the stage categories can be found in folder gen_dk in the CSV file named stage_categories.

Example	Comment
[	Begin list
{	Begin object
"Edge": [	Begin edge
{	
"CategoryToStage": "Insert_unique_edge_ID_here"	Begin new edge
}	End new edge
},	
"Insert_stage_category_ID",	Find the ID from gen_dk CSV library
"Insert_stage_ID_number"	Insert stage ID number
]	End edge
}	End object
]	End list

Chapter 5

The import function in LCAbyg

5.1 Import a json project in LCAbyg

When importing json files in LCAbyg it is recommended to open the program using LCAbyg.exe. LCAbyg is found in the same folder as the LCAbyg program **C:\Program Files\SBi\LCAbyg 5 (64 bit) (5.2.1.0)***.

Once LCAbyg is opened, you simply click on "Fil" and then "Ny" and select your project from the list, see the example in Figure 5.1.It is possible to modify the building project in LCAbyg after it is imported from the json files.

**The last number is the version number and change when you download newer versions of the program.*

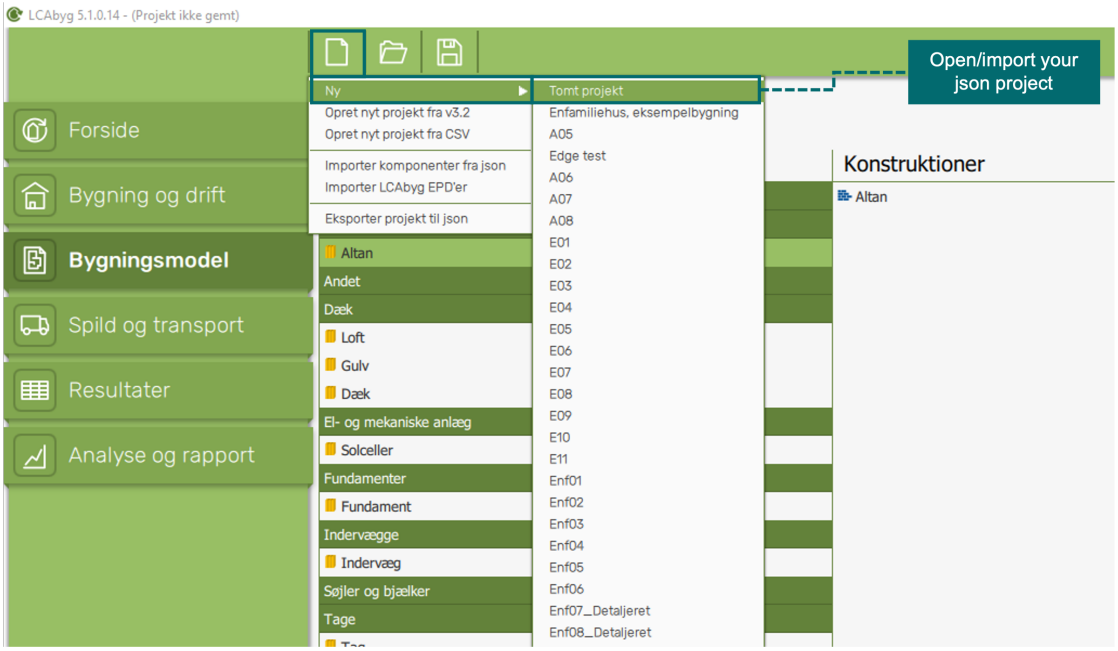


Figure 5.1: Import a json project in LCAbyg

5.1.1 Check the code

To check if the code is correct, you must do the following steps, see Figure 5.2.

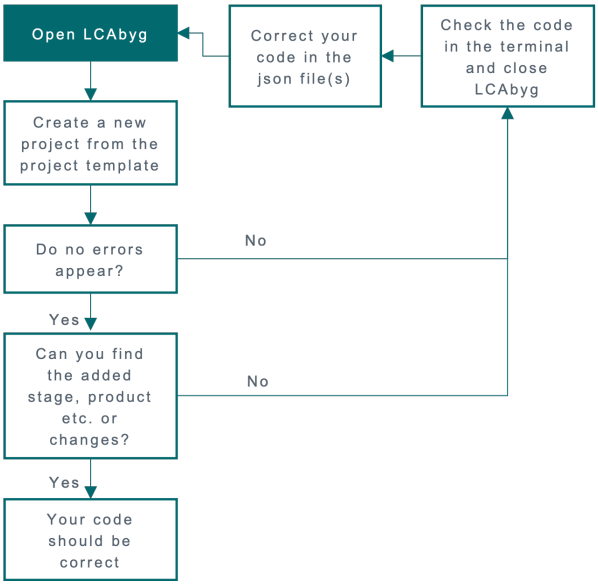


Figure 5.2: How to check your json code. See Appendix D for troubleshooting

5.2 Import json components in LCAbyg

Once you have opened a project in LCAbyg you can import a folder containing json components. As a minimum, the folder must contain an "element.json"-file as described in Section 4.2.9. Please note that the folder must not contain components with the same id.

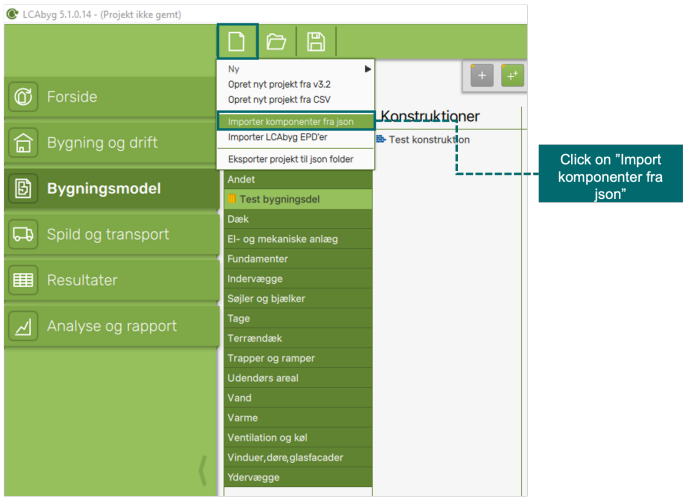
It is recommended to open the program using LCAbyg.exe. LCAbyg is found in the same folder as the LCAbyg program **C:\Program Files\SBi\LCAbyg 5 (64 bit) (5.2.1.0)***. Read more about how to use the LCAbyg.exe in Appendix D.1.1

**The last number is the version number and change when you download newer versions of the program.*

Please follow the next steps if you want only to import json components and not a complete json project.

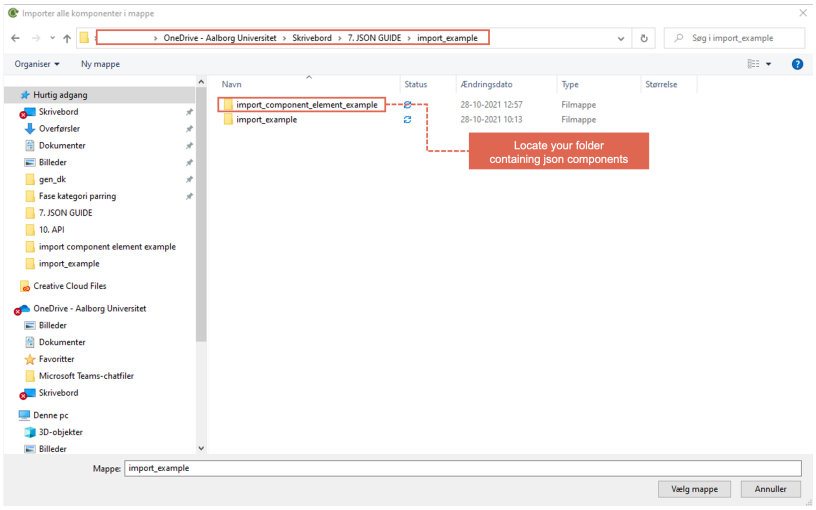
Step 1 - Go to "Importer komponenter fra json"

Click on the menu "Fil" and then "Importer komponenter fra json".



Step 2 - Locate the folder

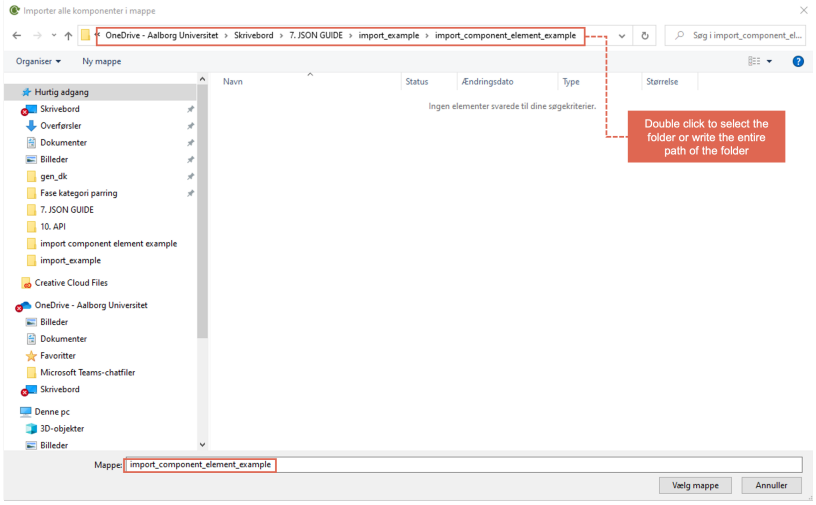
Select the path or locate your folder containing the json components you want to import, see the example below. See the folder “import_component_element_example” for an example of a simple folder structure that only contains one new element. See Appendix B.2 on how to create more nodes and edges for constructions, products, and phases.



Step 3 - Double click to select the folder

The image below shows exactly how the folder should appear when selecting it for import. If the folder is not selected correctly, then LCAbyg will not be able to work.

Note: If the folder may appear to be empty this should just be ignored as it does not influence the import-function.



Appendix A

Detailed Overview

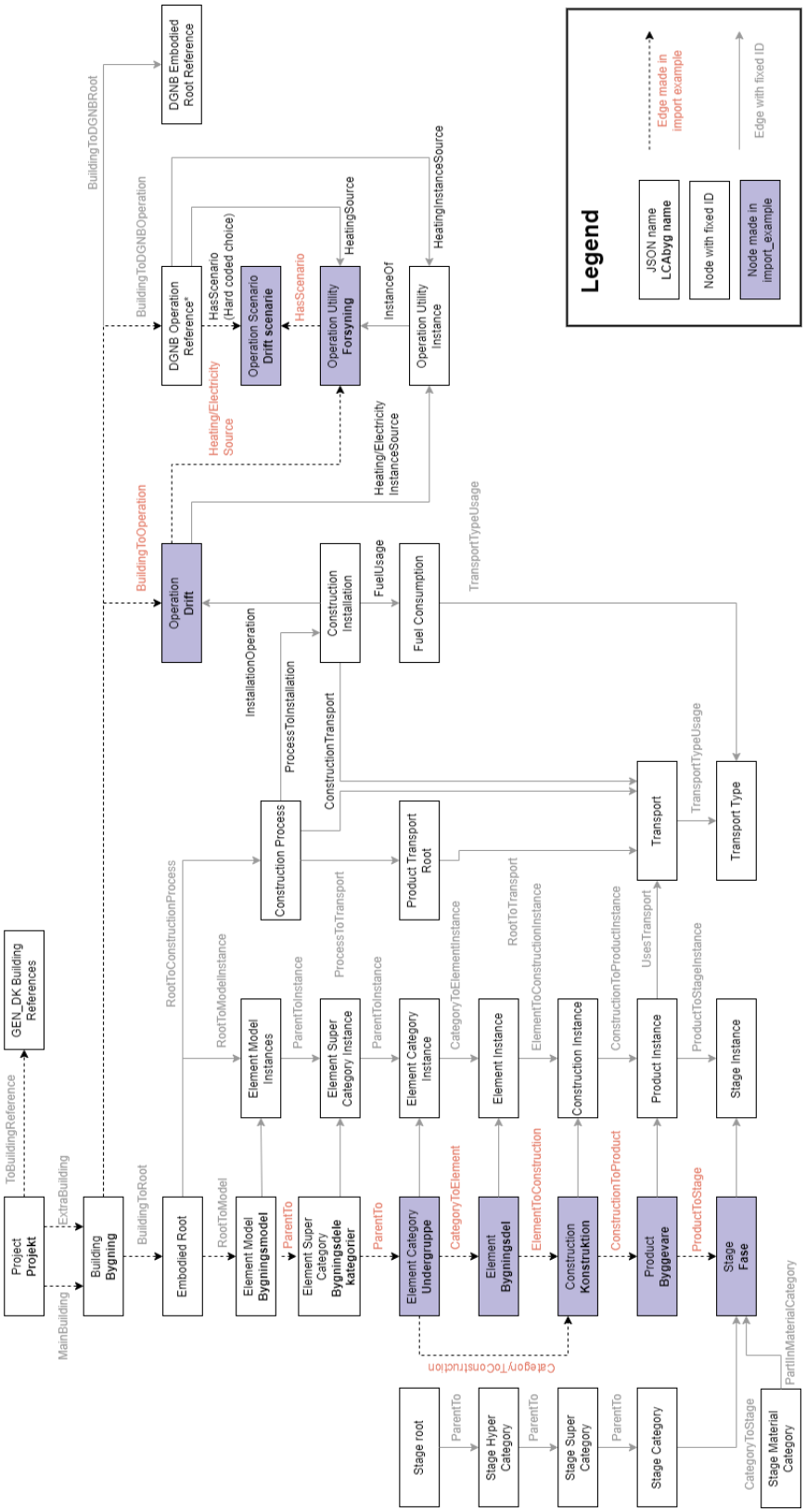


Figure A.1: Detailed overview of the how LCAbyg is structured in json

Appendix B

JSON files (import_example)

B.1 The Building Operation and Energy Consumption


```
[
  {
    "Node": {
      "Operation": {
        "id": "0338d31e-3876-440d-a88c-2daa2dd81942",
        "electricity_usage": 17.3,
        "heat_usage": 0.0,
        "electricity_production": 0.0
      }
    }
  },
  {
    "Edge": [
      {
        "HeatingSource": "6b4e569e-cc28-49f4-9f3a-fb042e66b158"
      },
      "0338d31e-3876-440d-a88c-2daa2dd81942",
      "e967c8e7-e73d-47f3-8cba-19569ad76b4d"
    ]
  },
  {
    "Edge": [
      {
        "ElectricitySource": "f6055900-f9b5-441f-b15e-6e79deac0224"
      },
      "0338d31e-3876-440d-a88c-2daa2dd81942",
      "e967c8e7-e73d-47f3-8cba-19569ad76b4d"
    ]
  }
]
```


Listing B.1: Operation.json


```
[
  {
    "Edge": [
      {
        "BuildingToOperation": "81305021-5571-4fbb-917b-b2f049d61420"
      },
      "dcb10ab8-621d-4640-bc9a-1f5c1964199a",
      "0338d31e-3876-440d-a88c-2daa2dd81942"
    ]
  }
]
```


Listing B.2: BuildingToOperation.json


```
[
  {
    "Node": {
      "OperationUtility": {
        "id": "9244ddc3-b353-48ed-9dec-2f1d4fe6120f",
        "name": {
          "English": "Test",
          "German": "Test",
          "Danish": "Test"
        },
        "util_type": "Electricity",

        "data_points": {
      "2020":{
        "ADPF": 1896.78,
        "GWP": 356.634,
        "POCP": 0.0236371,
```

```
        "EP": 0.0999654,
        "SER": 0.0,
        "SENR": 0.0,
        "PER": 1111.0,
        "PENR": 2157.82,
        "ADPE": 0.00146681,
        "AP": 0.378718,
        "ODP": 3.79874e-12
      }
        }
      }
    }
  }
]
```

Listing B.3: OperationUtility.json

```
[
  {
    "Node": {
      "OperationScenario": {
        "id": "8c83dda7-0d26-44b1-9acb-d67b85b6e0d4",
        "name": {
          "Danish": "Test",
          "English": "Test",
          "German": "Test"
        },

        "description": "Test"
      }
    }
  }
]
```

Listing B.4: OperationScenario.json

```
[
  {
        "Edge": [
            {
                "HasScenario": "c98e8554-103b-4b63-85c1-611be7cae33b"
            },
            "9244ddc3-b353-48ed-9dec-2f1d4fe6120f",
            "8c83dda7-0d26-44b1-9acb-d67b85b6e0d4"
        ]
    }
]
```

Listing B.5: HasScenario.json

B.2 The Building Model


```
[
  {
    "Node": {
      "Project": {
        "id": "29267fbd-a31d-40b8-a904-c37e21538e05",
        "name": {
          "Danish": "Test eksempel"
        },
        "address": "Testvej 1, 1111 Testbyen",
        "owner": "Test",
        "lca_advisor": "Test",
        "building_regulation_version": "BR2018"
      }
    }
  }
]
```


Listing B.6: Project.json


```
[
  {
    "Node": {
      "Building": {
        "id": "dcb10ab8-621d-4640-bc9a-1f5c1964199a",
        "scenario_name": "Original bygningsmodel",
        "locked": "Unlocked",
        "description": {
          "English": "Test",
          "German": "Test",
          "Danish": "Test",
          "Norwegian": "Test"
        },
        "building_type": "Other",
        "heated_floor_area": 0.0,
        "gross_area": 0.0,
        "integrated_garage": 0.0,
        "external_area": 0.0,
        "gross_area_above_ground": 0.0,
        "storeys_above_ground": 0,
        "storeys_below_ground": 0,
        "storey_height": 0.0,
        "initial_year": 2020,
        "calculation_timespan": 50,
        "calculation_mode": "Normal",
        "outside_area": 0.0,
        "plot_area": 0.0,
        "energy_class": "LowEnergy"
      }
    }
  }
]
```


Listing B.7: Building.json


```
[
  {
    "Edge": [
      {
        "MainBuilding": "8a6f5c9c-8ad1-48f2-9991-74acb58c4e82"
      },
      "29267fbd-a31d-40b8-a904-c37e21538e05",
      "dcb10ab8-621d-4640-bc9a-1f5c1964199a"
    ]
  }
```

```
]
```

Listing B.8: MainBuilding.json

```
[
  {
    "Edge": [
      {
      "BuildingToRoot": "48f2019b-8954-41d6-b6d2-2a9f3f2334d2"
      },
      "dcb10ab8-621d-4640-bc9a-1f5c1964199a",
      "216cf5d6-3e9d-43ec-b0d8-5aee02240c28"
    ]
  }
]
```

Listing B.9: BuildingToRoot.json

```
[
  {
    "Node": {
      "Element": {
        "id": "e2a8a510-d492-4d10-8bd5-ffca8e47b152",
        "name": {
          "Danish": "Test element",
          "English": "",
          "German": ""
        },
        "source": "User",
        "comment": "",
        "enabled": true,
        "excluded_scenarios": []
      }
    }
  }
]
```

Listing B.10: Element.json

```
[
  {
    "Node": {
      "EmbodiedRoot": {
        "id": "216cf5d6-3e9d-43ec-b0d8-5aee02240c28"
      }
    }
  },
  {
    "Edge": [
      {
        "RootToModel": "fc48c913-a28f-4bdf-a6f3-fa9d6124f957"
      },
      "216cf5d6-3e9d-43ec-b0d8-5aee02240c28",
      "aeefab8a-e825-4d14-b0c3-e87b7759e5b2"
    ]
  },
  {
    "Edge": [
      {
        "RootToConstructionProcess": "a8a3fa25-694a-4b93-8a0a-bd73a8fb1d7a"
      },
      "216cf5d6-3e9d-43ec-b0d8-5aee02240c28",
      "349738da-9747-4d5c-b508-2a810317166f"
    ]
  }
```

```
]
```

Listing B.11: EmbodiedRoot.json

```
[
  {
    "Node": {
      "Construction": {
        "id": "45f040ac-158e-4bcf-87bc-795a6dfbc2d9",
        "name": {
          "Danish": "Test konstruktion",
          "English": "Test construction"
        },
        "unit": "M",
        "source": "User",
        "comment": "Test",
    "locked": true
      }
    }
  }
]
```

Listing B.12: Construction.json

```
[
  {
    "Edge": [
      {
        "ElementToConstruction": {
          "id": "6e095089-b2c3-4f35-9a31-e57923ca8cae",
          "amount": 59.9,
          "enabled": true,
          "excluded_scenarios": []
        }
      },
      "e2a8a510-d492-4d10-8bd5-ffca8e47b152",
      "45f040ac-158e-4bcf-87bc-795a6dfbc2d9"
    ]
  }
]
```

Listing B.13: ElementToConstruction.json

```

[
  {
      "Edge": [
        {
          "CategoryToConstruction": {
            "id": "bb4be81f-7779-4884-9d8c-050f02edd3d7",
            "layers": [
              1
            ]
          }
        },
        "10a52123-48d7-466a-9622-d463511a6df0",
        "45f040ac-158e-4bcf-87bc-795a6dfbc2d9"
      ]
    }
]
```

Listing B.14: CategoryToConstruction.json

```
[
  {
    "Edge": [
```

```
      {
      "ConstructionToProduct": {
        "id": "0ff4b687-6c6e-4007-8bd8-46ac93925697",
        "amount": 0.44,
        "unit": "M3",
        "lifespan": 120,
        "demolition": false,
        "delayed_start": 0,
        "enabled": true,
        "excluded_scenarios": []
      }
      },
      "45f040ac-158e-4bcf-87bc-795a6dfbc2d9",
      "87b88977-bea0-4a70-9508-b4b49b20085a"
    ]
    }
]
```


Listing B.15: ConstructionToProduct.json


```
[
  {
    "Node": {
      "Stage": {
        "id": "b70da4f1-bfea-410c-888b-5ee358bcd36f",
        "name": {
          "English": "Test phase (A1-A3)",
          "German": "Testphase (A1-A3)",
          "Danish": "Test fase (A1-A3)"
        },
    "hyper_category": "Mineral building products ",
        "comment": "",
        "source": "User",
        "locked": true,
        "valid_to": "2022-01-01",
        "stage": "A1to3",
        "stage_unit": "M3",
        "indicator_unit": "M3",
        "stage_factor": 1.0,
        "mass_factor": 1800.0,
        "indicator_factor": 1.0,
        "external_source": "Ökobau.dat 2020 II",
        "external_id": "dea7df16-f59b-4842-a66c-cb9463a58ae3",
        "external_version": "20.19.120",
        "external_url": "http://www.oekobaudat.de/OEKOBAU.DAT/resource/processes/dea7df16-f59b
    -4842-a66c-cb9463a58ae3?version=20.19.120",
        "data_type": "Generic",
        "compliance": "A1",
        "indicators": {
          "ADPF": 1896.78,
          "GWP": 356.634,
          "POCP": 0.0236371,
          "EP": 0.0999654,
          "SER": 0.0,
          "SENR": 0.0,
          "PER": 1111.0,
          "PENR": 2157.82,
          "ADPE": 0.00146681,
          "AP": 0.378718,
          "ODP": 3.79874e-12
        }
      }
    }
  }
]
```


Listing B.16: Stage.json

```
[
 {
   "Edge": [
     {
       "ProductToStage": {
         "id": "ef2adc40-4ee0-415c-a5f8-c16e60100fee",
         "excluded_scenarios": [],
         "enabled": true
       }
     },
     "87b88977-bea0-4a70-9508-b4b49b20085a",
     "b70da4f1-bfea-410c-888b-5ee358bcd36f"
   ]
 }
]
```


Listing B.17: ProductToStage.json


```
[
  {
    "Edge": [
      {
      "CategoryToStage": "9de0f451-f67a-449b-b1fb-e8fc32379b7d"
      },
      "2fb2b8ec-ac21-4d1d-85c9-98243e7c5c56",
      "b70da4f1-bfea-410c-888b-5ee358bcd36f"
    ]
    }
]
```


Listing B.18: CategoryToStage.json

Appendix C

Choices in Building node

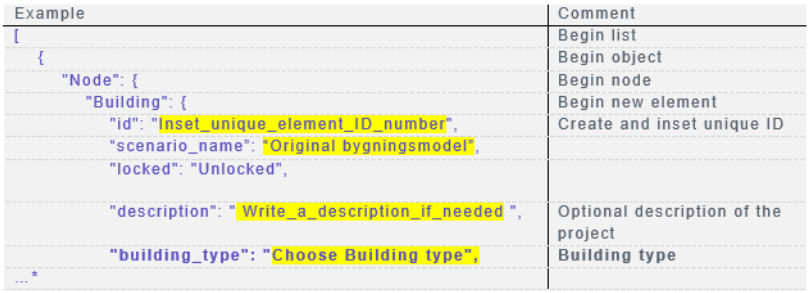
C.1 Building type (bygningstype)

Currently, ten building types exist in LCAbyg. The ten types are described in Table C.1.

Table C.1: Building types

Type	Description
Office	Offices
School	Schools and other institutions
ResidentialBuildingSingleFamilyHouse	Residential, single family housing
ResidentialBuildingMultiStoreyBuilding	Residential, multi story building
ResidentialBuildingTerracedHouse	Residential, terraced housing
Store	Stores
Logistics	Logistic building
Production	Production building
Hotel	Hotels and hostels
Other	Buildings that don't fall into any of the building types

The building type ("Byningstype") can be set in the JSON file named Building, see the code snippet below.



Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"Building": {	Begin new element
"id": "Inset_unique_element_ID_number",	Create and inset unique ID
"scenario_name": "Original bygningsmodel",	
"locked": "Unlocked",	
"description": " Write_a_description_if_needed ",	Optional description of the project
"building_type": "Choose Building type",	Building type
...*	

*The full json file for the Building can be found in Appendix B.2 or in import_example folder.

C.2 Calculation mode (beregningstype)

Currently, three calculation modes exist in LCAbyg. The three modes are described in Table C.2.

Each of the three calculation methods are connected to the Edges ElectricitySource and HeatingSource. If DGNB is chosen, then the DGNB Operation Reference with electricity supplement ("El tillæg") and heat supplement "Varme tillæg") is generated automatically by LCAbyg.

Table C.2: Calculation modes

Mode	Description
BR23	Bygningsreglement 2023 rules
Normal	Normal LCAbyg rules
SC	Sustainability Class (Den Frivillige Bæredygtighedsklasse)

The calculation mode ("Beregningstype") can be set in the JSON file named Building, see the code snippet below.

Example	Comment
[	Begin list
{	Begin object
"Node": {	Begin node
"Building": {	Begin new element
"id": "Inset_unique_element_ID_number",	Create and inset unique ID
"scenario_name": "Original bygningsmodel",	
"locked": "Unlocked",	
"description": " Write_a_description_if_needed ",	Optional description of the project
"building_type": "Choose Building type",	Building type
"heated_floor_area": Write integer value,	Inset an integer
"gross_area": Write integer value,	Inset an integer
"gross_area_above_ground": Write integer value,	Inset an integer
"storeys_above_ground": Write integer value,	
"storeys_below_ground": Write integer value,	
"storey_height": Write decimal value,	Format 0.0
"initial_year": Write year as integer value,	Format yyyy
"calculation_mode": "Chose mode"	**"BR23", "Normal", "DGNB", or "SC"**
...*	

*The full json file for the Building can be found in Appendix B.2 or in import_example folder.

Appendix D

Troubleshooting

D.1 Debug information in LCAbyg

D.1.1 LCAbyg.exe (debug tool)

The file LCAbyg_debug.exe is a version of LCAbyg that also contains a window that shows logs and error messages. This mode is useful for debugging errors related to the import of json files in LCAbyg.

Step 1 - Find LCAbyg.exe

The program is located together with LCAbyg in **C:\Program Files\SBi\LCAbyg 5 (64 bit) (5.3.1.0)***, see Figure D.1.

**The last number is the version number and change when you download newer versions of the program.*

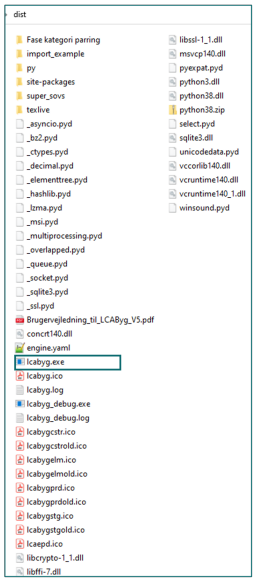


Figure D.1: Location of LCAbyg.exe

Step 2 - Open LCAbyg.exe

Double click the file to open the program. The terminal opens and after approximately 5-10 second LCAbyg programs opens automatically. You can now proceed to open the json project as described earlier.

Note: LCAbyg does not work if you place the curse or write inside the terminal.

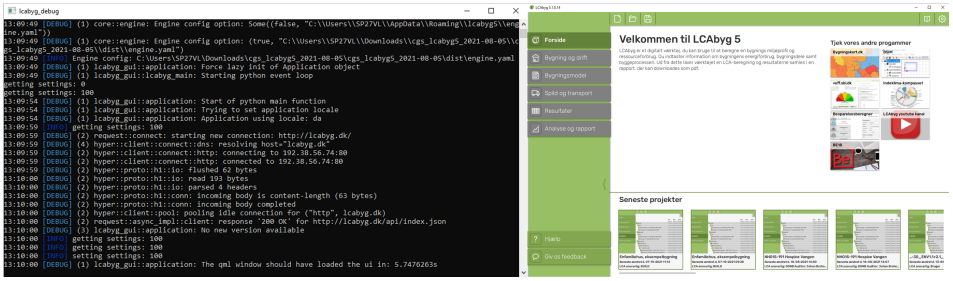


Figure D.2: Opening LCAbyg in debug mode via the file “LCAbyg.exe

Step 3 - If you can't open LCAbyg.exe

Go to the file explorer where LCAbyg_debug is located and type cmd and click enter This will open a terminal that indicate errors in your engine.yalm file.

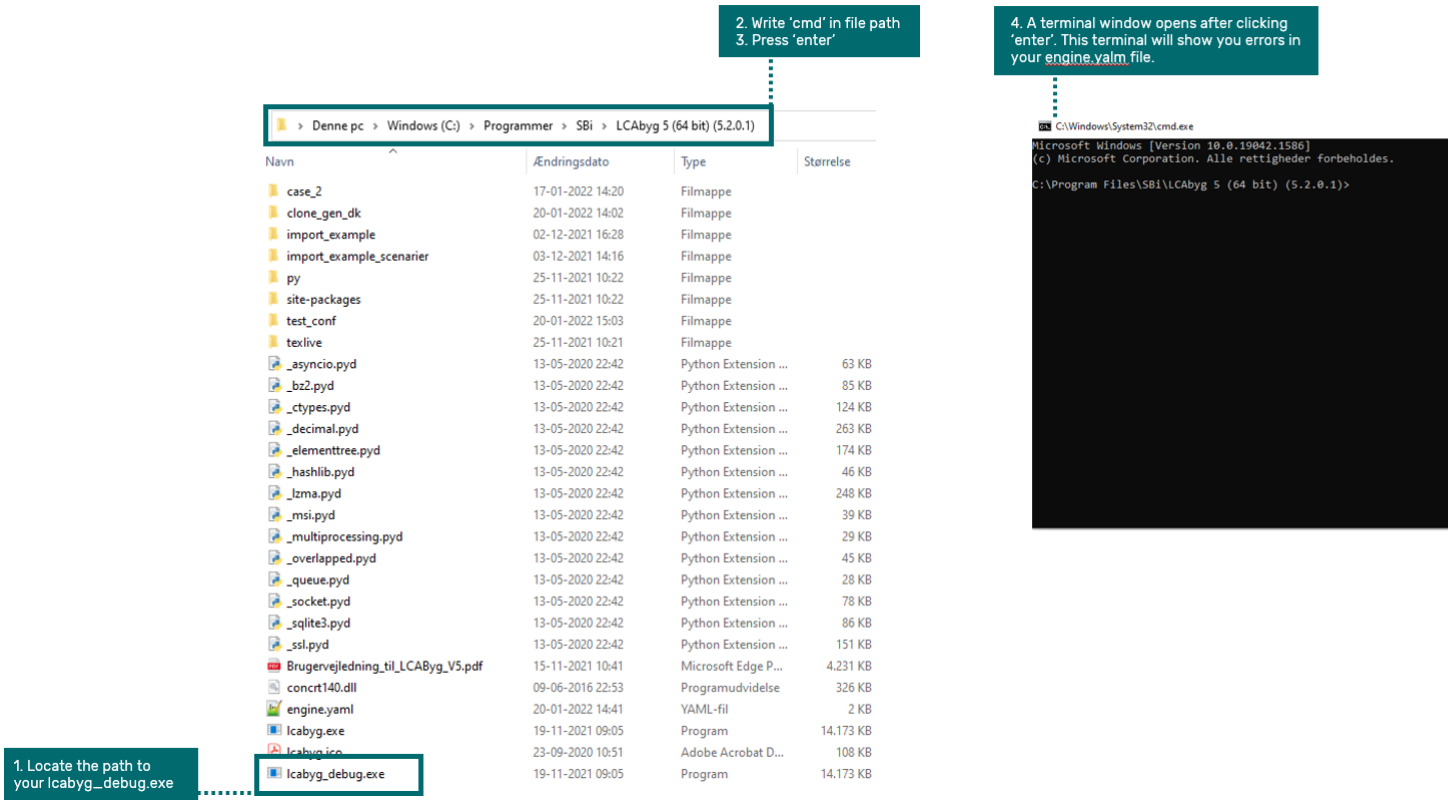


Figure D.3: Opening terminal when it is not possible to open lcabyg_debug.exe

Appendix E

ID external source

E.1 ID from external source

Following screenshot shows an example on an external source (Ökobau, Reinforcing steel) used when creating a new stage

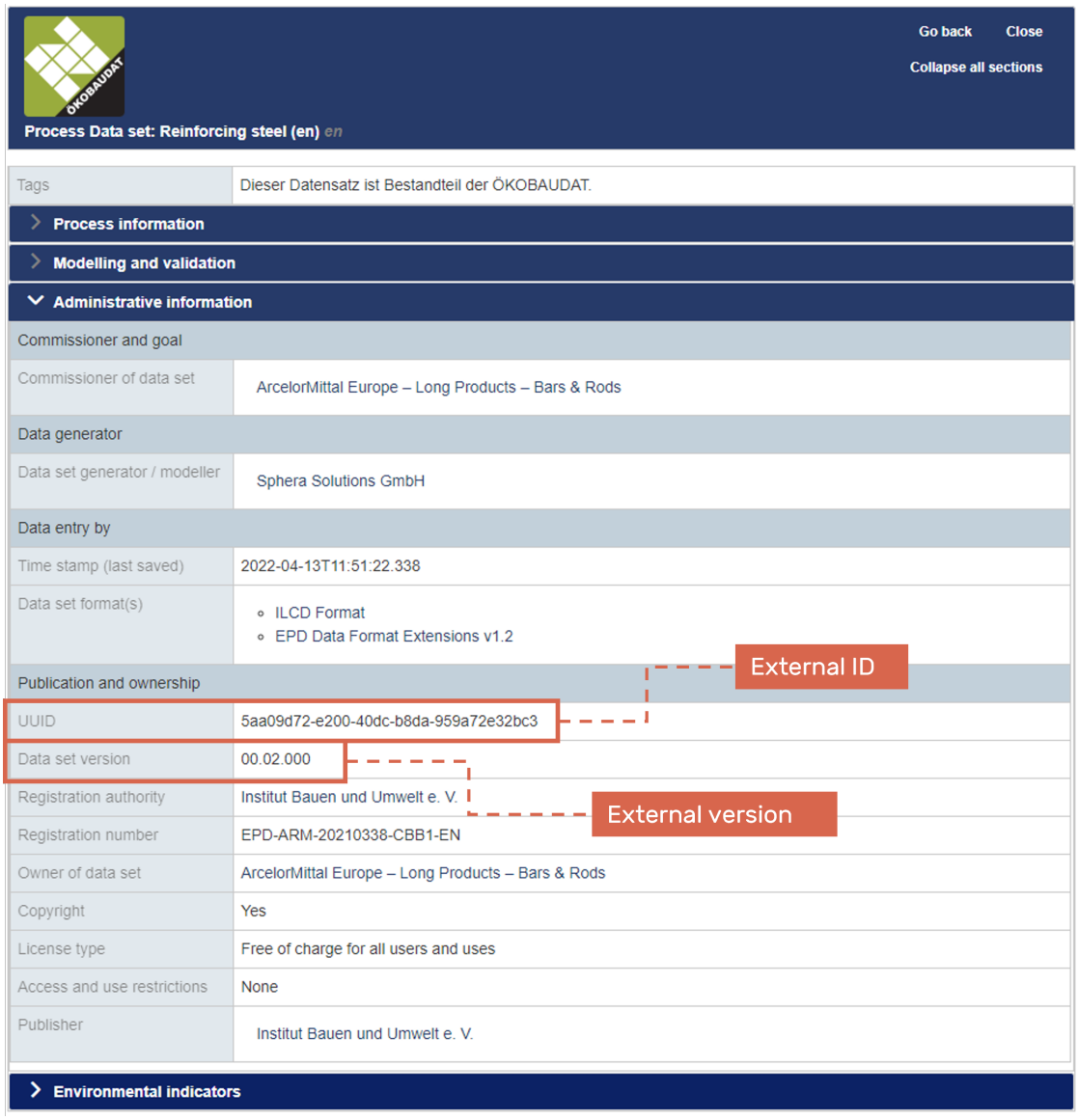


Figure E.1: Example on ID and version of external source