Illustrative Prototype
for Home Energy Model Building Specification Input Documentation

Version 1.0
12 September 2024

Chris Gordon-Smith
chris.gordon-smith@foe.co.uk

Table of Contents

1. Introduction

1.1. Purpose of Document

This document provides an Illustrative Prototype for documentation to describe the Building Specification Input to the UK government's Home Energy Model (HEM). This input enables a user to set the various options and parameter values defining the characteristics of the building(s) being modelled. A document describing how to set-up the input would, in effect, be a "driver's manual" for the Home Energy Model (in more technical language, an input interface specification).

This Illustrative Prototype is not itself such a document. Instead, it provides an example of the nature of material needed in such a document and a suggested approach to structuring it. It is hoped that this will help towards the production of full documentation.

1.2. Background

In December 2023 the UK government published a consultation for the Home Energy Model. The purpose was to "seek views on the new Home Energy Model which will replace the Standard Assessment Procedure (SAP) for the energy rating of homes."

Friends of the Earth responded to this consultation.[1] The response was very favourable overall, particularly on the open-source approach. However, it did raise concerns. These were in two main areas, the use of the open-source code as the ultimate legal reference for the HEM methodology, and some aspects of the approach to wrappers. These concerns are documented in detail in the consultation response document. As part of describing these concerns, the need for HEM core interface documentation was highlighted.

1.3. Motivation

At Friends of the Earth we are exploring how the Home Energy Model might enable development of a tool or tools to help inform our supporters and others on insulation and heating options. The Illustrative Prototype has been produced as part of this activity. We hope that it will also be useful as an example for others producing HEM documentation.

Any comments or other input on this document will be very welcome. This includes comments on the inferences documented in Section 5 or answers to the questions raised in that section.

Please send any input to chris.gordon-smith@foe.co.uk.

1.4. Document Scope and Content

This document focuses on the Building Specification Input[2] for the Home Energy Model. Weather data (EPW or CIBSE) can be input separately to HEM. These weather inputs are not covered by this document. They can be download from publicly available websites.

The following outlines the content of this document:

[1] The consultation response can be found in the Openmod Modelling Forum

[2] The term building specification is used in the file hem.py in the Home Energy Model GitHub repository and is also adopted here.

- Section 1 (this section) provides introductory material, including the background, purpose of and motivation for the document.
- Section 2 identifies limitations of the Illustrative Prototype.
- Section 3 introduces the JSON (JavaScript Object Notation) format that is used for the Building Specification Input.
- Section 4 describes data modelling concepts and graphical conventions used in this document. The Building Specification Input is a large data structure containing many objects and sub-objects. Data modelling techniques are used to describe this structure using diagrams, tables and explanatory text.
- Section 5 is the core content. It shows the overall structure of objects in the input and identifies the detailed properties (name/value pairs) of these objects.
- Appendix A: MIT License includes the MIT License, as is required for the MIT Licensed material included in this document.

1.5. About the Author

Chris Gordon-Smith is a retired IT Professional with wide experience of IT including system modelling, simulation and data modelling.

1.6. Acknowledgements

This document was created as part of volunteer activity with Friends of the Earth. The author thanks Mike Childs and Toby Bridgeman at Friends of the Earth for their input and support.

2. Limitations of the Illustrative Prototype

2.1. Partial Scope

In its current form the Illustrative Prototype covers five out of the eighteen top level objects that make up the HEM Building Specification Input. There is much more to be done!

2.2. Illustrative Prototype is Based on Inferences

There is currently no published documentation describing the Building Specification Input interface. Therefore much of the material in this document has necessarily been inferred from the HEM program code[3] and the demonstration input (JSON) files. A certain amount of 'detective work' is involved and it is often not possible to reach a definitive description of a data item, or to be sure that assumptions made are correct. This document must be read with this in mind. Much of it represents the author's best guess about the input options HEM provides and what it requires in the input. Comments on or corrections to the assumptions will be very welcome.

[3] Downloaded on 5 September 2024

3. JSON Features in the HEM Building Specification Input

The HEM Building Specification Input is provided to the Home Energy Model as a JSON (JavaScript Object Notation) formatted file. A useful introduction to JSON can be found on the JSON website.[4] The following outlines the main JSON features used in this document.

3.1. JSON Objects

The HEM Building Specification Input is formatted as a hierarchy of JSON *objects*. A JSON object is an unordered set of *name / value* pairs.[5] Its scope in the input is delimited by curly braces. '{' denotes the start and '}' denotes the end.

3.2. Names

A name is a *string* that identifies the value in a name/value pair. A string is a series of characters enclosed in double quotes, e.g. "start_day". Each name is followed by a colon (':') that separates it from its value.

3.3. Values

A value in the HEM Building Specification Input can be:

- A string in double quotes
- A number
- One of: true, false or null (unquoted)
- An object
- An array

3.4. Properties

Name/value pairs are also called *properties*.

3.5. Arrays

An *array* is an ordered collection of values. It is delimited by square brackets. It begins with '[' and ends with ']'. Values in the array are separated by commas (',').

3.6. Example

Figure 3 shows an object named "ApplianceGains". This contains two objects named "lighting" and "cooking". The "lighting" object has properties named "start_day", "time_series_step", "gains_fraction", "EnergySupply" and "schedule". The value associated with "schedule" is an object containing an array of numbers. The name of this array is "main".

4. Data Modelling Concepts and Graphical Conventions

This section presents the data modelling concepts and graphical conventions used in Section 5 to show the structure and content of the HEM Building Specification Input. Attention is drawn to Object Naming Types and Object Roles which are introduced as a means of distinguishing different kinds of object in the input.

[4] See also Standard ECMA-404: The JSON Data Interchange Syntax

[5] The term *key* is often used instead of name.

4.1. Object Naming Types

JSON objects in the input can be named in different ways as follows:

- *HEM Named Object:* This is the most common type of naming. The object must be named according to the requirements of the HEM system
- *User Named Object:* These objects have names that must be assigned by the user
- *Anonymous Object:* These objects do not have names

4.2. Object Roles

In the case of objects for which the name is not (yet) known (Anonymous Objects and User Named Objects) it is useful to identify the object according to its *role*. An object's role indicates its purpose or responsibility in the context of the overall input structure.

4.3. Object Relationships

The HEM Building Specification Input includes a hierarchical structure in which lower-level objects are *part of* (contained in) a higher-level object. Most of the relationships in the data model are of this kind.

In some cases one object refers to another object without containing it. This indicates a *dependency* or a *works with* relationship.

4.4. Graphical Conventions

The diagrams in this document use the following graphical conventions

4.4.1. Representation of Objects

A box represents a JSON object.[6]

4.4.2. Representation of Object Naming Type and Name / Role

Objects in the HEM Building Specification are named in different ways. The placing and style of text in a box indicates the naming type:

- *HEM Named Object:* If a box has a text string in CapitalisedWords at the centre (for example ApplianceGains in
- Figure 2) then the box denotes a HEM Named Object and the text string is its name.
- *User Named Object*: If a box has a text string at the centre in italic snake case (for example *appliance_gains_category* in
- Figure 2) then the box denotes a User Named Object and the text string is its role. The actual name will be specified by the user.
- *Anonymous Object:* If a box has a text string in italic snake case at the top (eg *shading item* in Figure 4) then it denotes an Anonymous Object and the text string is the object's role.

[6] In strict data modelling terms it would be more correct to say that a box represents a *class* (or type) of object. However, repeatedly making this distinction explicit could lead to cumbersome text and it is simpler to leave it implicit. The meaning should be clear from the context. Where there is a need to refer to an actual object rather than a class, the term object *instance* is used.

4.4.3. Representation of Relationships

Relationships are represented by lines between the related objects. A line with a filled diamond at one end indicates that the object at the other end is *part of* the object at the diamond end. Looking at the relationship in the other direction, it can also be called *composition*.

A dotted line with an arrow at one end indicates a dependency relationship in which an object refers to the object at the arrowhead end.

4.4.4. Representation of Cardinality

At each end of a relationship there may be zero, one or more instances of each of the JSON objects.[7] This indicates the relationship's *cardinallty*. A '1' by an object indicates that there is one and only one instance of the object related to an instance at the other end of the relationship line. The text '0..1' by an object indicates that there may or may not be an instance of the object related to an object instance at the other end of the relationship line. More generally, text such as 'n..m' (eg '8..36') next to an object indicates that there must be between n and m instances of the object related to an object instance at the other end of the relationship line. '*' is a wildcard meaning any positive number, so that '0..*' next to an object means that it may have any number of instances related to an object instance at the other end of the relationship line, including zero.

4.4.5. Arrays

Arrays are not objects and are not explicitly shown on the diagrams. An array can be a value within an object. The values within an array can often themselves be objects and in such cases the *array item* objects are shown on the data model diagram. See for example Figure 4**: ExternalConditions Sub-Hierarchy**.

4.5. Input Data Model vs HEM Internal Data Model

The data model diagrams in this document describes the structure of the data *input* to HEM. In many cases this will be similar to the internal structure of the HEM system data. However, it should be noted that there can be differences. For example, in the HEM Building Specification Input, InternalGains and ApplianceGains are separate objects, but in the HEM core (file project.py) ApplianceGains are added in to become part of the InternalGains object.

[7] See the footnote in Section A.4.4.1.

5. HEM Building Specification Input Structure and Content

Figure 1 shows the top-level structure of the HEM Building Specification Input. Diagram conventions are as explained in Section 4.

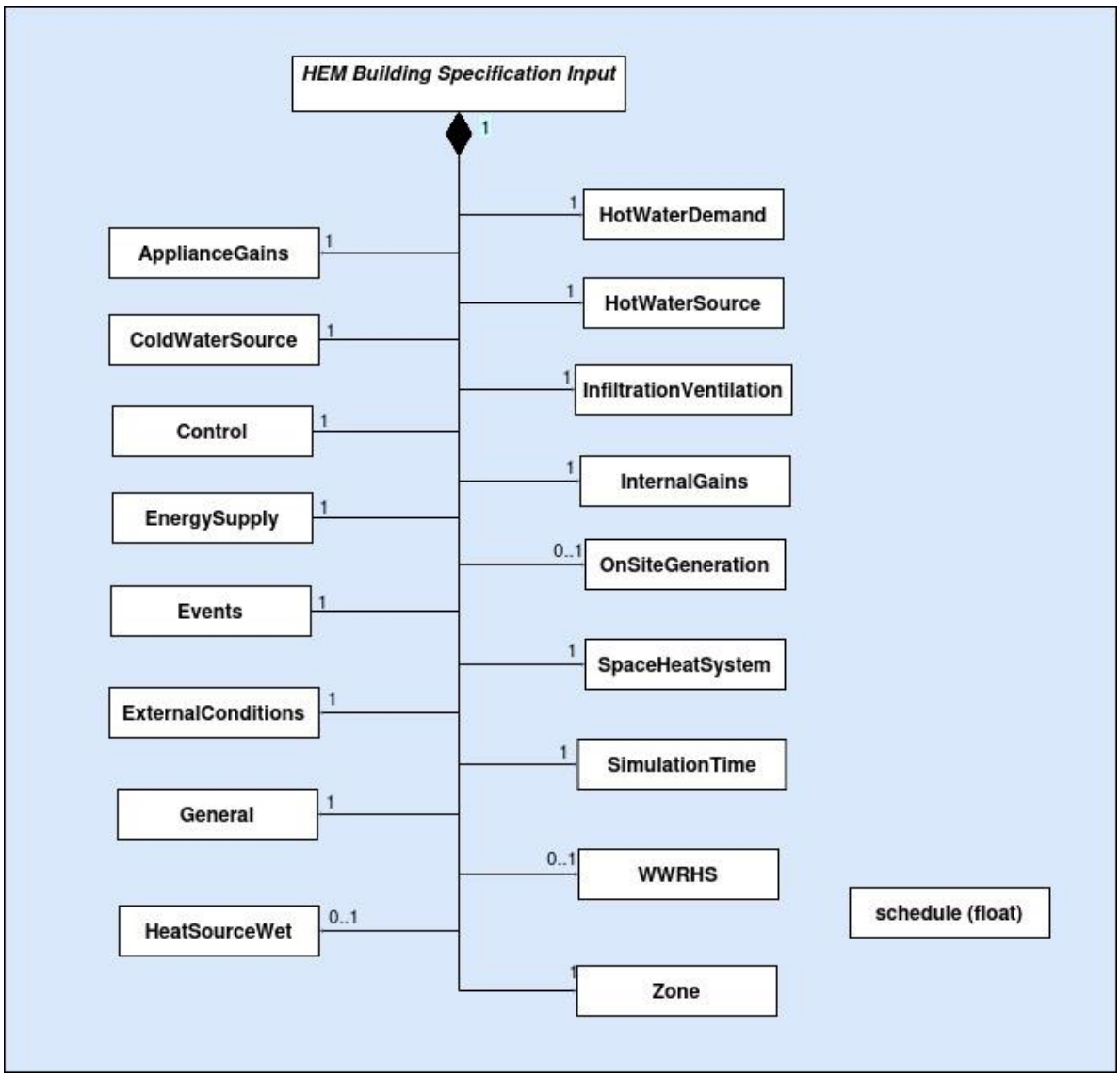

Figure 1: HEM Building Specification - Top Level Structure

The main structure is hierarchical, with a single HEM Building Specification Input (anonymous) object at the top.[8] This top-level object is composed of a set of sub-objects, many of which have their own internal sub-hierarchies (shown below for some of the objects). The object at the top of a hierarchy is called the *root* object. Figure 1 shows all of the sub-objects in the level immediately below the single HEM Building Specification Input root object.

In most cases the HEM Building Specification Input has exactly one sub-object of each type. This is indicated by a number '1' next to the sub-object. In three cases the text '0..1'

[8] Technical Note: For diagram readability the relationship lines are merged. This is possible without loss of information because they are all *'part of'* relationships with a filled diamond at the HEM Building Specification Input end.

appears instead of '1'. This means that the sub-object is optional. For example, the HEM Building Specification Input object may or may not contain an OnSiteGeneration object.

The diagram also includes a schedule (float) object. This is outside the main hierarchy (it is not part of any other object) but it is referred to by other objects (
Figure **2** shows an example of this).

The following sub-sections focus on the various (sub-)objects that make up the HEM Building Specification Input object. In this Illustrative Prototype only a few of the sub-objects are included. Each sub-section includes:

- A diagram showing the part of the overall hierarchy (the sub-hierarchy) focused on the sub-object under consideration (the focus object)
- An example of JSON code providing input for the focus object. The code snippets are from the file demo.json provided under the MIT License on the UK Government's Home Energy Model GitHub site. A copy of the MIT License is included in Appendix A as required.
- Tables showing properties (name/value pairs) of the focus object and each of the lower level objects in the sub-hierarchy. For each property the following are shown:
    - Name
    - Value type
    - Example value(s)
    - Number of occurrences of the property (related to cardinality)
    - Notes

Ideally the tables would include column(s) stating the meanings and allowed values for all data items in the input file. However, definitive information on this cannot be included because although in many cases inferences can be made from the program code or demonstration input, it is very often not possible to do so with certainty.[9]

A Notes column is however included in each table indicating what has been inferred so far and raising questions. Examples include:

- Inference: The words "lighting" and "cooking" in the ApplianceGains input are not keywords expected by HEM. They are user named categories of heat gains from appliances in the building. They could be anything that the user condiders appropriate.
- Question: The demonstration files show that allowed types for a shading_item include "obstacle" in the case of the building as a whole, and "overhang" in the case of a window. Can the HEM building as a whole have an overhang? Can a HEM window have an obstacle?

Corrections to the inferences and answers to the questions will be very welcome (see Section 0). Clarification on the value types will also be useful, particularly in cases where a value could be either an integer or a floating point number.

5.1. ApplianceGains Sub-Hierarchy

The ApplianceGains sub-hierarchy can be used to specify heat input to the dwelling from appliances.
Figure 2 shows the structure of the ApplianceGains hierarchy. Figure 3 shows example JSON input for ApplianceGains.

[9] The same applies to the Value type column that is included in each table.

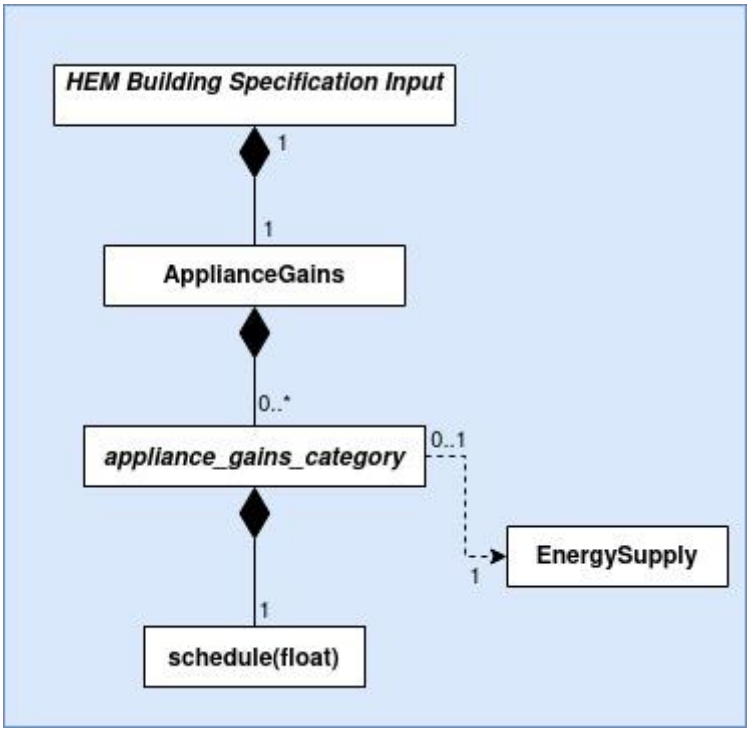

Figure 2: ApplianceGains Sub-Hierarchy

Figure 2 shows that the ApplianceGains root object is at the top of the ApplianceGains sub-hierarchy. This sub-hierarchy may include a number of user named *appliance_gains_category* objects. Each of these specifies heat input in a category such as lighting or cooking and must include a schedule(float) object indicating how the heat input varies over time. An *appliance_gains_category* object must also identify an EnergySupply object indicating the source of energy for appliances in the category. The EnergySupply object is not part of the ApplianceGains hierarchy. It is a separate independent object.

```
"ApplianceGains": {
        "lighting": {
                "start_day": 0,
                "time_series_step": 1,
                "gains_fraction": 0.5,
                "EnergySupply": "mains elec",
                "schedule": {
                        "main": [32.0, 46.0, 33.0, 21.0, 12.0, 17.0, 25.0, 46.0]
                }
        },
        "cooking": {
                "start_day": 0,
                "time_series_step": 1,
                "gains_fraction": 1,
                "EnergySupply": "mains elec",
                "schedule": {
                        "main": [300.0, 120.0, 220.0, 750.0, 890.0, 150.0, 550.0, 280.0]
                }
        }
},
```

Figure 3: Example ApplianceGains input

5.1.1. ApplianceGains Object

An ApplianceGains object is a HEM Named Object that specifies the names of *appliance_gains_category*s, each of which is a type of appliance gain. Table 1 shows the format and content of the ApplianceGains object.

Table 1: ApplianceGains *o*bject format and content

Name	Value Type	Num. Occurrences	Example(s)	Inferences and Questions
User defined name for a category of appliance gains	appliance_gains_category	0..*	Typical names are: "lighting", "cooking"	Inference: Name is the appliance_gains_category name. There can be multiple categories and so this property can be repeated for different categories. The cardinality is accordingly shown as 0..*.

5.1.2. appliance_gains_category object

An *appliance_gains_category* object is a User Named Object that specifies a particular type of appliance heat gain. For example, heat gains associated with cooking. The object includes a schedule (float) object specifying the amounts of heat provided at different times. Table 2 shows the format and content of the object.

Table 2*: appliance_gains_category* object

Name	Value Type	Num. Occurrences	Example	Inferences and Questions
"start_day"	int	1	0	Question: The file internal_gains.py says "first day of the time series, day of the year, 0 to 365 (single value)". But what is the time series?
"time_series_step"	float	1	1	

“gains_fraction”	float	1	0.5	
“EnergySupply”	string	1	“mains elec”	Inference: Identifies an EnergySupply” object that appears in the HEM Building Specification Input
“schedule”	schedule(float) object	1	See Section 5.4.	

5.2. ExternalConditions Sub-Hierarcy

The External Conditions sub-hierarchy specifies external conditions at the building location. For example, external air temperatures, wind speeds, and shading.

Figure 4 shows the structure of the sub-hierarchy.

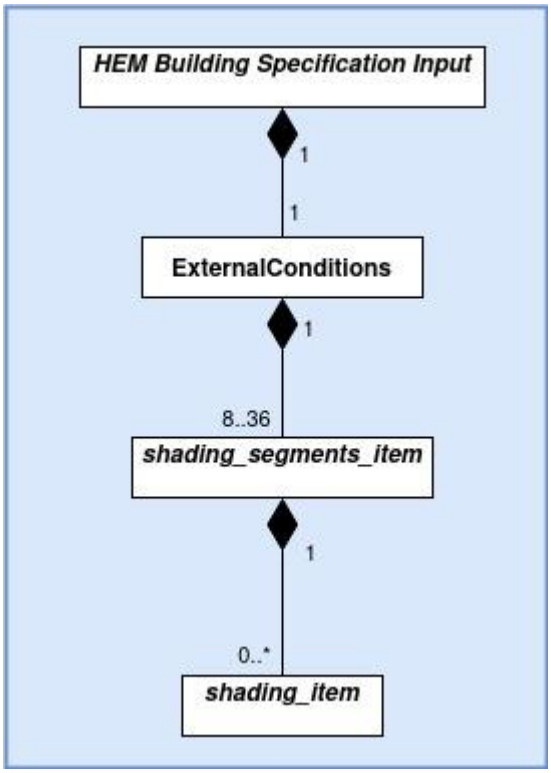

Figure 4: ExternalConditions Sub-Hierarchy

The ExternalConditions object includes a shading_segments array containing from 8 to 36 shading_segment_items. Each of these items is an object that identifies a segment of the ground plane and may contain a shading array. If present, the shading array has one or more shading_item objects, each of which specifies height and distance for a physical object near the building.

Figure 5 shows example JSON input for ExternalConditions.

```
"ExternalConditions": {
        "air_temperatures": [0.0, 2.5, 5.0, 7.5, 10.0, 12.5, 15.0, 20.0],
        "wind_speeds": [3.9, 3.8, 3.9, 4.1, 3.8, 4.2, 4.3, 4.1],
        "diffuse_horizontal_radiation": [0, 0, 0, 0, 0, 0, 0, 0],
        "direct_beam_radiation": [0, 0, 0, 0, 0, 0, 0, 0],
        "solar_reflectivity_of_ground": [0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2],
        "latitude": 51.42,
        "longitude": -0.75,
        "timezone": 0,
        "start_day": 0,
        "end_day": 0,
        "time_series_step": 1,
        "january_first": 1,
        "daylight_savings": "not applicable",
        "leap_day_included": false,
        "direct_beam_conversion_needed": false,
        "shading_segments": [
            {"number": 1, "start360": 0, "end360": 45},
                {"number": 2, "start360": 45, "end360": 90},
                {"number": 3, "start360": 90, "end360": 135},
                {"number": 4, "start360": 135, "end360": 180,
                        "shading": [
                                {"type": "obstacle", "height": 10.5, "distance": 12}
                        ]
                },
                {"number": 5, "start360": 180, "end360": 225},
                {"number": 6, "start360": 225, "end360": 270},
                {"number": 7, "start360": 270, "end360": 315},
                {"number": 8, "start360": 315, "end360": 360}
        ]

},
```


Figure 5: Example ExternalConditions input

5.2.1. ExternalConditions Object

An ExternalConditions object is a specifier for a range of environmental and other conditions at the building. The object includes an array of shading_segments_items. Table 3 shows the format and content of the object.

Table 3: ExternalConditions Object

Name	Value Type	Num. Occurrences	Example	Notes
"air_temperatures"	array[float]	1	[0.0, 2.5, 5.0, 7.5, 10.0, 12.5, 15.0, 20.0]	Inference: Assume that the number of array entries is the number of hours between “start” and “end” in the Si,mulationTime object.
"wind_speeds"	array[float]	1	[3.9, 3.8, 3.9, 4.1, 3.8, 4.2, 4.3, 4.1]	Ditto
"diffuse_horizontal_radiation"	array[float]	1	[0, 0, 0, 0, 0, 0, 0, 0]	Ditto
"direct_beam_radiation"	array[float]	1	[0, 0, 0, 0, 0, 0, 0, 0]	Ditto
"solar_reflectivity_of_ground"	array[float]	1	[0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2]	Ditto
"latitude"	float	1	51.42	
"longitude"	float	1	-0.75	
"timezone"	int	0..1	0	Inference: Assume this is ignored. It is set to 0 by the HEM code (See __init__ function for the Project class)
"start_day"	int	0..1	0	Inference: Assume ignored as for “timezone”. Question: The file external_conditions.py says “first day of the time series, day of the year, 0 to

Name	Value Type	Num. Occurrences	Example	Notes
				365 (single value)". But what is the time series? Is it related to weather?
"end_day"	int	0..1	0	Inference: Assume ignored as for "timezone".
"time_series_step"	float	0..1	1	Inference: Assume ignored as for "timezone".
"january_first"	int	0..1	1	Inference: Assume ignored as for "timezone".
"daylight_savings"	string	0..1	"not applicable"	Inference: Assume ignored as for "timezone".
"leap_day_included"	bool	0..1	false	Inference: Assume ignored as for "timezone".
"direct_beam_conversion_needed"	bool	1	false	
"shading_segments"	array[shading_segments item], size 8 to 36	1	See Figure 5	Inference: Num. occurrences is based on comments in external_conditions.py.

5.2.2. shading_segments_item

A shading_segments_item is an object that specifies a segment of the ground plane. If there are physical objects in the segment shading the building then the shading_segments_item includes an array (named shading) that specifies these physical objects.

Table 4 shows the format and content of a shading_segments_item.

Table 4: shading_segments_item format and content

Name	Value Type	Num. Occurrences	Example	Notes
”number”	int	1	3	
“start360”	float (degrees)	1	90	Question: Is this a bearing from due North for the start of the segment?
“end360”	float (degrees)	1	135	Question: Is this a bearing from due North for the end of the segment?
“shading”	array[shading item]	0..1		

5.2.3. shading_item

A shading item object specifies a physical object that produces shade. Table 5 shows its format and content.

Table 5: shading item format and content

Name	Value Type	Num. Occurrences	Example	Notes
“type”	string	0..*	“obstacle”	Question: The demonstration files show that allowed types for a shading_item include “obstacle” in the case of the building as a whole, and “overhang” in the case of a window. Can the building as a whole have an overhang? Can a window have an obstacle?
“height”	float	1	10.5	
“distance”	float	1	12	

5.3. InternalGains Sub-Hierarchy

The InternalGains sub-hierarchy specifies heat gains from non-appliance sources[10] inside the building. Figure 6 shows the overall structure.
Figure 7 shows example JSON input for InternalGains.

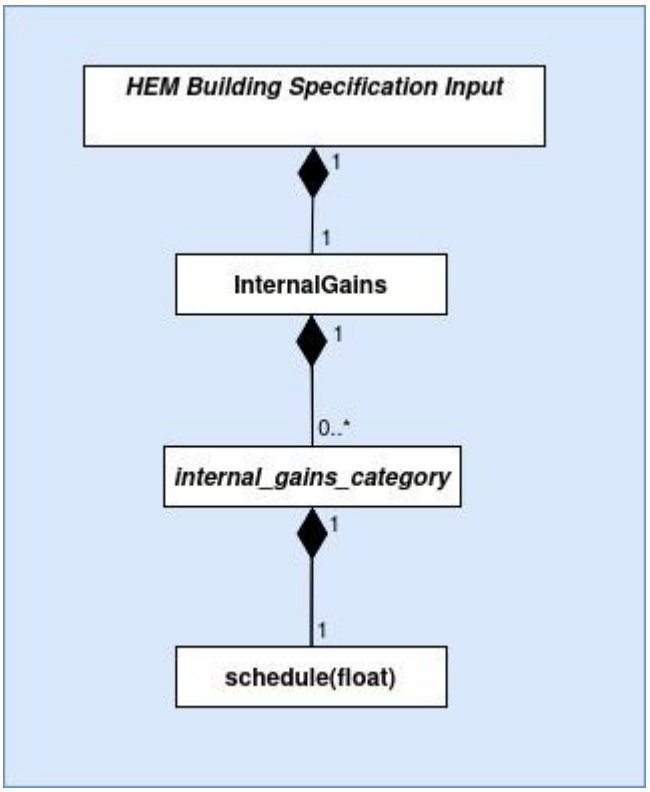

Figure 6: InternalGains Sub-Hierarchy

```
"InternalGains": {
        "metabolic gains": {
                "start_day": 0,
                "time_series_step": 1,
                "schedule": {
                        "main": [256, 368, 584, 416, 712, 448, 816, 648]
                }
        },
        "other": {
                "start_day": 0,
                "time_series_step": 1,
                "schedule": {
                        "main": [88, 200, 184, 376, 576, 544, 728, 608]
                }
        }
},
```

Figure 7: Example InternalGains input

[10] Heat gains from appliances are input separately in the "ApplianceGains" object. However, they are added internally by the HEM into the InternalGains. Therefore within HEM, ApplianceGains become part of the InternalGains object.

5.3.1. InternalGains Object

An InternalGains object specifies the names of a set of internal_*gains_category*s, each of which is a type of internal gain. Table 6 shows its format and content.

Table 6: InternalGains object

Name	Value Type	Num. Occurrences	Example	Notes
User defined name (in double quotes) for a category of internal gains.	internal_gains _ category	0..*	"metabolic gains"	Inference: "metabolic gains" is a user assigned name for a category of internal gains.

5.3.2. internal_gains_category Object

An *internal_gains_category* object specifies a particular type of internal gain. For example, metabolic gains associated with people in the building. The object includes a schedule(float) object specifying the amounts of heat provided at different times. Table 7 shows the format and content of the object.

Table 7: internal_gains_category object

Name	Value Type	Num. Occurrences	Example	Notes
"start_day"	integer	1	0	Question: File internal_gains.py says that this is the "start of the time series". What data series does this refer to?
"time_series_step"	integer	1	1	Question: File internal_gains.py says that this is the timestep of the time series data in hours. Must the number of hours be an extact number of days? Question: How does this relate to

				time_series_step in ApplianceGains? Can the two be different?
"schedule"	schedule(float) object	1	See Section 5.4	Question: How does this relate to the "schedule" in the ApplianceGains object. Must the two time series be aligned, or can they each have a different set of times?

5.4. schedule (float)

A schedule (float) is an object containing a set of floating point values associated with a series of times. This object can be used in multiple places in the HEM Building Specification Input object hierarchy.

Table 3.8: schedule (float)

Name	Value Type	Num. Occurrrences	Example	Notes
"main"	array[float]	1	[88, 200, 184, 376, 576, 544, 728, 608]	Inference: HEM provides generic capability for schedule data structures to be held internally. However, the schedule objects in the HEM Building Specification Input covered by this document always use the 'float' schedule type and represent only the 'main' schedule. See schedule.py and project.py.

5.5. SimulationTime Sub-Hierarchy

The SimulationTime sub-hierarchy specifies simulation time-stepping requirements. Figure 8 shows its relationship with the HEM Building Specification Input object. Figure 9 shows example JSON input for the SimulationTime object. Table 8 shows its format and content.

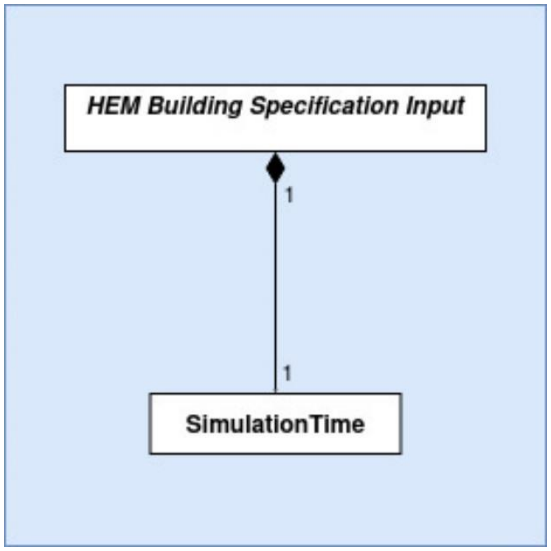

Figure 8: SimulationTime sub-hierarchy

```
"SimulationTime": {
        "start": 0,
        "end": 8,
        "step": 1
},
```

Figure 9: Example SimulationTime input

Table 8: SimulationTime object

Name	Value Type	Num. Occurrences	Example	Notes
“start”	float	1	0	Inference: This maps to starttime in the HEM internal SimulationTime object.
“end”	float	1	8	Inference: This maps to endtime in the HEM internal SimulationTime object.
“step”	float	1	1	

Appendix A: MIT License

The JSON code snippets in this document are from the UK Government's Home Energy Model published under the MIT License. The license text is included below.